



TO ALL WHOM IT MAY CONCERN:

- 5 ~~BE IT KNOW THAT~~ Coach K Wei, a citizen of the People's Republic of China,
having a residence at 1612 Worcester Road, #606A, Framingham, Massachusetts 01702
and, Zakir Magdum, a citizen of the United States of America, having a residence at 20E
Farmington Drive, Shewsbury, 01720 have invented a certain new and useful
METHODS AND TECHNIQUES FOR DELIVERING RICH JAVA
10 **APPLICATIONS OVER THIN-WIRE CONNECTIONS WITH HIGH**
PERFORMANCE AND SCALABILITY.

ABSTRACT

A system is developed for real time delivery of rich Java applications over thin wire with high performance and extreme scalability. The benefits of the system are:

1. No code modification is required for existing Java applications.
 2. High performance computing over low bandwidth or long latency network connections;
 3. Compatible with existing Java application server environments;
 4. "Develop Once, Deploy Anywhere": the same Java application can run over many different kind of client devices from PC to PDA to Handheld computers, without re-coding;
 5. Java applications running on this system will be extremely scalable. A single server can handle thousands of users.
- This document provides a sample implementation of the system. A presentation and communications architecture for enabling highly interactive, real-time, rich networked applications is presented. This architecture enables network applications to be as rich, interactive as locally installed client/server applications without the heavy downloading/installation requirements that come with such client/server applications. Using this approach, applications can be delivered to a browser, a remote desktop environment, or a wireless edge device, with transmitting only a few kilobytes of data, but still perform equally well as client/server applications. Importantly, this approach enables existing Java applications to be used without modification while delivering the above mentioned benefits. Additional benefits include enabling the same application to run from various devices in an adaptive fashion that delivers the best user experience in each specific device environment.

TITLE OF THE INVENTION

**METHODS AND TECHNIQUES FOR DELIVERING RICH JAVA
APPLICATIONS over**

OVER THIN-WIRE CONNECTIONS WITH HIGH PERFORMANCE and
scalability AND SCALABILITY

FIELD OF THE INVENTION **Cross Reference to Related Applications**

5 This application claims the benefit of U.S. provisional application Serial No. 60/256,594
 filed on 12/19/2000 and entitled METHODS AND TECHNIQUES FOR DELIVERING
RICH JAVA APPLICATIONS OVER THIN-WIRE CONNECTIONS WITH HIGH
PERFORMANCE AND SCALABILITY which is commonly assigned and the contents
 of which are expressly incorporated herein by reference.

10

Field of the Invention

— This invention relates to computing architecture, and more particularly to a
 method and apparatus for running applications business logic on a backend server, with
 the presentation layer of the application handled on the client device, and
 15 communications between the presentation layer and business logic layer via messages
 based on formats such as XML.

DETAILED DESCRIPTION OF THE INVENTION

Background of the Invention

20 The rapid growth of the web, including the Internet and the wireless web, brings
 tremendous challenges as well as opportunities. Though web-centric computing seems to
 be the inevitable future, there are great challenges not yet answered which include:

- The capability to support compelling applications within unpredictable network environments.
- 25 • The overwhelming complexities of web application development and deployment.
- The rapid growth and divergence of client environments including web browsers, PDAs, Handheld computers, and so on.

Nexaweb Technologies Inc's flagship product, Nexel, is Accordingly, there is a need for
 an enabling platform that, for the first time, enables developers to write programs for one
 30 platform and deploy them anywhere—anywhere, i.e., to any device in any environment—

and empowers environment, while empowering these programs with unmatched richness and performance even over thin network connection.

Summary of the Invention

5 In general, in one aspect, the invention features a method for delivering one or more applications over a network in which the business logic of the application is running on the backend server, and the user interface of the application is rendered on a client-device who is connected to the backend server via a network. The Graphics User Interface API and event processing API of the application are implemented to be network-aware instead
 10 of being local machine centric as traditional GUI APIs. The method includes the following steps. Running an application on the backend server. The application in turn invokes GUI API to present its user interface, however, the network-aware GUI API is invoked. Next, the network aware GUI API translates the application's presentation layer information into a pre-determined format based messages which describes the Graphical
 15 User Interface, event processing registries and other related information. Such information describes the presentation layer of the application in a high level, object level, which minimizes network traffic. Next sending such messages to the client device via a network. Next, processing the messages and rendering the user interface by a client-side program, which delivers the best possible user experience for that device according
 20 to the capability of the specific client device. Next, transmitting necessary user input and client-side events back to the server by the client-side program via a predetermined protocol. Next, processing the user input and client-side events on the backend server, translating such events and inputs as if they were locally generated, and sending such translated events and inputs to the application for processing. Next, encoding and routing
 25 the output of the application to the client device using the predetermined messaging format. Finally, further processing the output by the client-side program to refresh the Graphical User Interface thereat.

30 Implementations of this aspect of the invention may include one or more of the following features. The Graphics User Interface API and event processing API may be Java Foundation Classes (including Swing, AWT and so on). The client-side program may be

a computer program based on Operating System's API, such as Windows API, X Windows API and so on. The client-side program may be a wireless device program written using the device's Operating System's API, such as Palm API and Windows CE API. The client-side program may be a Java program written using Java API. The JAVA API may be AWT, Personal Java, Java 2 Micro Edition based GUI API or Java Swing. The predetermined protocol may be HTTP, HTTPS, WAP or a proprietary protocol. The predetermined messaging format may be based on XML or a proprietary format. The network may be the Internet, a wireless network, or a local area network. The local area network may be a bandwidth-limited slow speed network. The client device is selected from a group including workstations, desktops, laptops, PDAs, wireless devices and other edge devices. The server and the client device may be combined into one entity.

In general, in another aspect, the invention features a server-side API based programming model for network programming, which frees or greatly simplifies the complexity of network programming by freeing developers from client-side issues. The presentation layer of the application is written using this server-side API. The business logic layer and data layer of the application is written using other appropriate server-side technologies. The supporting infrastructure of this server-side API sends the application's user interface information to the client-side device for presentation, handles communications problems, renders the application's user interface and dispatches necessary user input events back to the server for processing.

In general, in yet another aspect, the invention features a method and a system for delivering existing Java applications over the network without modification of the application's code and without downloading the application to the client side. The system re-implements standard Java GUI APIs such as AWT and Swing into a network-aware implementation without changing the APIs, enabling existing Java applications to run on this network-aware GUI API without modifications. The Java application runs completely on the server-side. The network-aware API translates and delivers the application's presentation information into short messages based on formats such as XML via a certain communication protocol. The system's client-side program that understands

- these messages interprets and renders the user interface of the Java applications, essentially produces the look and feel of the application as if the entire application is running on the client device. The client program further interacts with the user, dynamically updates the user interface and sends necessary user inputs back to server for
- 5 processing. The system's server program receives such user inputs, translates them into Java compatible user inputs, such as Java events, and further routes such user inputs to the Java application for processing. The output of the Java application's processing is sent to the system's client program, which updates the user interface of the application.
- 10 In general, in yet another aspect, the invention features a method and a system for delivering the same application over some network to multiple devices, maximizing the user experience of each device by best leveraging the specific capability of each device, without rewriting the application specifically for each device. The system runs the application on the server side. The system's server-side program translates and delivers
- 15 the presentation information of the application into messages based on selected format such as XML. Such messages contain high level description of the application's user interface. Such high level, instead of pixel level or graphics primitive level description, gives sufficient flexibility in interpretation without losing the gist of the information. Specific client-side programs are built for each specific client device leveraging the
- 20 special features of each device. This client-side program interprets these messages and renders the user interface of the applications in a way that is best optimized for the client device, delivering the best user experience possible on that specific client device. The client-side program accepts user inputs, update the user interface, and sends necessary user inputs back to the server. The system's server program receives such user inputs,
- 25 translates them into application compatible user inputs, and further routes such user inputs to the application for processing. The output of the application's processing is sent to the system's client program, which updates the user interface of the application accordingly.
- 30 Among the advantages of this invention may be one or more of the following. No code modification is required for existing Java applications. High performance computing over

low bandwidth or long latency network connections. Compatible with existing Java application server environments. "Develop Once, Deploy Anywhere": the same Java application can run over many different kind of client devices from PC to PDA to Handheld computers, without re coding. Java applications running on this system will be extremely scalable. A single server can handle thousands of users.

The details of one or more embodiments of the invention are set forth in the accompanying drawings and description below. Other features, objects and advantages of the invention will be apparent from the following description of the preferred embodiments, the drawings and from the claims.

Brief Description of the Drawings

Referring to the figures, wherein like numerals represent like parts throughout the several views:

15

FIG. 1 is an overview diagram of the Nexel Platform architecture;

FIG. 2 is a block diagram of the Nexel Server;

20 FIG. 3 is a block diagram of the Nexel Server interfacing with a Servlet Engine via the Nexel Servlet;

FIG. 4 is a block diagram of the Nexel Server interacting with a standard Java application;

25

FIG. 5 is a block diagram of the Nexel Swing components;

FIG. 6 is a block diagram of the Nexel Layout manager;

30 FIG. 7 is an overview diagram of an enterprise-scale deployment of the Nexel Platform;

FIG. 8 is an overview diagram of the Nexel Client Kernel architecture;

FIG. 9 is a block diagram of the Client component hierarchy;

5 FIG. 10 is a block diagram of the Nexel classes including core classes (package com.nexaweb.core), package com.nexaweb.net, the layout manager and additional classes;

FIG.11 is a block diagram of the package com.nexaweb.plaf.ce;

10

FIG. 12 is a block diagram of the package com.nexaweb.plaf.pc; and

FIG. 13- FIG. 20 depict an example an implementation of the Nexel Core Classes.

15 Detailed Description of the Invention

~~This document is a design specification for Nexel Server for Java.~~ Nexel Server (or, Nexel Presentation Server) is the core of the Nexel Platform. The functionalities of Nexel Server for Java are:

- 20 • To manage and execute Java applications on the server;
- To adapt the ~~UI~~User Interface (UI) of these Java applications according to the capability of the client environment, such as screen size and client computing power.
- To deliver the User Interface (UI) of these Java applications to the client
- 25 computer over network;
- To dispatch user events from the client device to corresponding applications.

The engineering goals of Nexel Server for Java are:

- Compatibility with existing applications: Nexel Server ~~will be~~is compatible with existing Java applications and ~~will be~~is able to run existing Java applications
- 30 without code modification;

- Zero learning curve: the Nexel Platform ~~will~~does not change how developers write programs and ~~will~~does not require developers to learn anything new. Existing Enterprise Java Bean (EJB) components, application server environments, database and so on can be used without change.
- 5 • Extreme scalability: each Nexel Server ~~will be~~is able to run several thousands of Java applications concurrently.
- High performance: The Nexel Platform ~~will~~can deliver native-like performance over any network connection.
- 10 • Compatibility with Java 2 Enterprise Edition (J2EE) Application Server environments: Nexel Server ~~will be~~is able to run together with J2EE Application Servers in a plug&play fashion. See white paper "The New Economy Enterprise Computing Architecture" for how Nexel Server and J2EE Application Server are complementary to each other in the new economy enterprise computing.

To achieve these goals, Nexel Server design takes the following approaches:

- 15 • A thread-based computing model (or, a service model). Each event handler in an application is invoked as a separate thread that dies after the event handling is finished. This is exactly the model that Java Servlet is using. Java Servlet has been used in major web sites for handling millions of hits a day. It is a proven scalable solution;
- 20 • Some of the state information ~~are~~is maintained on the client side. This releases server from maintaining all the information in memory;
- Multiple Java Virtual Machines (JVM) are utilized for large scale computing. These JVMs can be on different machines. As a result, they form a server farm that can scale to any requirement.

25

Nexel Server Kernel conforms to the Java Servlet API Specification. It can run within a web container from any J2EE application server.

30

1. Nexel Platform Architecture

Nexel brings an innovative approach to application delivery. Applications are developed using Java and standard J2EE platform. Once these applications are deployed with the Nexel Delivery Server, they can be used anywhere on the web. The NexWebNexel
 5 Delivery Server will present application UI on the client machine. The general architecture view of the Nexel Platform 100 is shown in Figure 1. Nexel consists of two parts. The server side part is called 120, that includes a "Delivery Server." Server" 122. Its main responsibility is to interact with the application and extract its UI and communicate with a client part. The client side part 110, that includes a "Client Kernel" is called
 10 "Client Kernel". It's 114. Its main responsibility is to recreate a applications UI an application UI 112 based on instructions from the server. Both these parts are described in detail in the following sections.

The Nexel server 122 works with any web server 124 that supports the a servlet interface.
 15 It runs on top of the J2EE platform 126, which can be run on the Linux, Windows NT, Linux 128, Windows NT 127, Unix 129, Solaris, and HP-UX etc. operating systems, among others. On the client side, we plan to develop players side 110, players may be included for playing these applications. These players will may be implemented on the Windows platform 102 with Internet Explorer 104 and Netscape 106 browsers. Also a
 20 proprietary player will be developed for Windows CE platform may be included in a Windows CE platform 103.

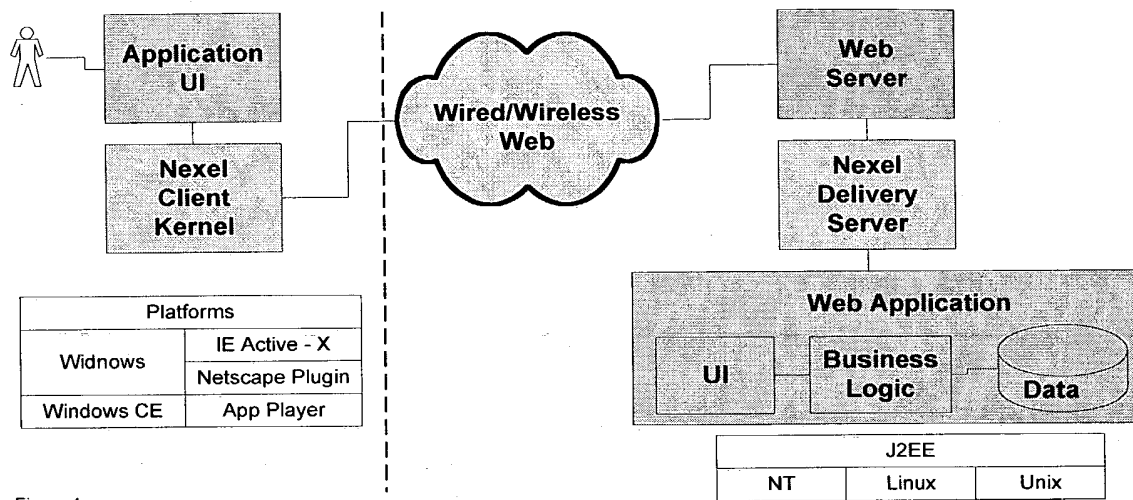


Figure 1

In addition to the delivery platform, NexaWeb also offers we also offer the following tools:

Development Tools	Layout Manager	
Deployment Tools		
Migration Tools	Code Analyzer	
Migration Tools	Code Analyzer	

5

Each of these deliverables is described in detail in following sections. To understand the function of all these parts it is important to understand how applications will be developed and deployed.

2. Application Development and Deployment Process

10

Nexel utilizes the existing process of developing applications for J2EE platform 126. In a three-tier architecture of an application 118 that includes data 115, business logic 116 and a UI 117, the data tier and business logic tier development process is remain unchanged. You can use any database or file system as data tier and you can develop the business logic using Enterprise Java Beans or any other Java technology or technology which can be called from Java. The UI layer has to be developed using Java Foundation Classes

15

(JFC) and Java. Many existing applications that use JFC for their UIs can be run in the Nexel environment without any modification. ~~NexaWeb will~~ We also produce migration tools for identifying problems in running existing code in the Nexel environment. The typical process for developing and deploying includes the following steps:

5

- a. Develop the data layer using enterprise databases and file systems.
- b. Develop business logic using enterprise java beans or using any other technology that can be called using Java.
- c. Develop the application UI using Nexel's JFC. You can use Layout Manager to
10 layout your screens at this point. Decide which events to process and connect the GUI Graphical User Interface (GUI) to business logic. At this point you can use Nexel's validation objects to process events on the client side.
- d. Install the Nexel application on the server using the Install Tool. You can customize the application UI for different client platforms using the Layout
15 Manager. You can save these customizations into different files.

3. Application Launching and Communication.

It is also important to understand how applications are launched in Nexel environment. ~~Communication~~ Referring to FIG. 2, communication between Delivery Server 122 and
20 Client Kernel 114 go through the Web server 124 using standard HTTP/HTTPS protocol 133. A socket-based communication ~~should~~ 135 may also be implemented for synchronous connection with the server. ~~An application developer should be~~ is able to choose the approach at development time. The sequence of events that occurs when a user wants to launch an application are:

25

1. User opens up the browser
2. User types in URL for the login HTML page. This page ~~three~~ includes two items
 - a. User Name
 - b. Password for the user.

Once the user enters the data, the form is submitted to the server. The server checks the user name and password and sends a page with available applications.

3. Each application has link setting with two query strings

- 5 a. Application Name
- b. User Name

The URL for each application will look like this

<http://websitename/UIServerServlet?Application=ApplicationName&User=UserName> .

10 Initially ~~only the web~~ Web servers that ~~support~~utilize servlet-programming models will be supported. ~~UI ServerServlet will~~ UI ServerServlet may be installed on the web server and will provide functionality to pass the message from client to UI Server. Once the user clicks on an application URL it is passed on to the server.

15 4. Once the Nexel server receives commands to launch an application it will go through the following steps

- a. It will instantiate the application.
- b. Nexel server will also detect the type of client, which is invoking of the application and pass on to the application.
- 20 c. A unique instance id will be assigned to application instance and pass on to the application.
- d. The application will run and it will talk to Nexel JFC API and create its UI. Nexel API will create a UI record format depending on the client type. At the end a record will be generated describing UI of the application.
- 25 e. Nexel server encrypts the record if secure communication is used.
- f. Nexel server will respond to the user with the UI record.

5. Once the Client Kernel receives the UI record it will go through the following steps

- 30 a. The client communication manager will read the UI record. It will decrypt the record if necessary.
- b. The client will read and parse the UI record.

- c. The client will convert the record into various UI commands and create the necessary components. Fill them with the data provided and setup event-monitoring mechanism.
 - d. The application is now displayed on the client screen
- 5 6. The ~~client~~Client Kernel will then monitor for all the events. When an event occurs it will find out whether it has to take any action. The action could be in two forms.
- a. Run the specified code on client side itself for that event. The code is specified using validation controls.
 - b. Notify the server of the event.
- 10 The Client Kernel will take necessary action depending on the Application. If the event has to be notified to the server then the event processing happens on the server side. The server goes through the following steps once it receives the notification.
- a. The server executes the method in the application, which is monitoring the
 - 15 events.
 - b. The server will monitor all the UI changes and converts them in UI record format.
 - c. It responds to the Client Kernel with that.
- 20 The client Kernel then goes through the process of updating UI. See steps 5.a through 5.d for details.
- 7. When user quits the application Client Kernel shuts down and notifies the server.
 - 8. When the server receives a message to quit it shuts down the application instance.

25 When Synchronous communication is a requirement, the UI server ~~will include~~includes a port number to which the client needs to establish the connection. Before creating any UI it connects to this port. Once the socket connection is established it is used for communicating later.

4. Nexel Delivery Server.

4.1 Platforms.

Nexel Server is implemented using Java technology. It ~~should support~~supports all the major web servers on major ~~platforms. Initially we will support~~platforms, including

- 5 Microsoft Internet Information Server (IIS) on Windows, Apache Web Server on Windows, Solaris, HP-UX, AIX, and Linux, Java Web Server on ~~Windows and Linux.~~
- Later on we will extend to other platforms and web servers.

Web Server	Initial Target Support	Phase 2 Target Support
Microsoft Internet Information Server	Windows NT/2000	
Windows, Solaris, AIX, and Linux, and		
Netseape Enterprise Server		Solaris, Windows, Linux, HP-UX, AIX
10 Netscape		
Apache Web Server	Windows, Linux, Solaris	HP-UX, AIX
Java Web Server	Windows, Linux, Solaris	Solarix, AIX

Enterprise Server on Solaris, Windows, Linux, HP-UX, and AIX.

4.2 Nexel Server Architecture

- 15 Nexel Server is ~~composed of four sub systems, as illustrated in Figure 1. There~~122
~~includes five sub systems 132, 134, 136, 138, and 140, shown in Figure 2. Their core~~
~~functionalities are shown in the following table 1. Note that two sub systems, Web Server~~
~~Adapter 132 and Java Servlet Engine 134, are standard and available freely. We do not~~
~~need to Implement these two subsystems in Phase 1 development. Open source code,~~
20 ~~such as Tomcat/Jakarta project from Apache Software Foundation, can~~may also be
~~used. We plan to modify these open source code by taking out the features that we don't~~
~~need. A simplified Java Servlet Engine, a simple Java Web Server, and adapters from IIS,~~
~~Apache, Netscape Web Server, are included as part of our product.~~

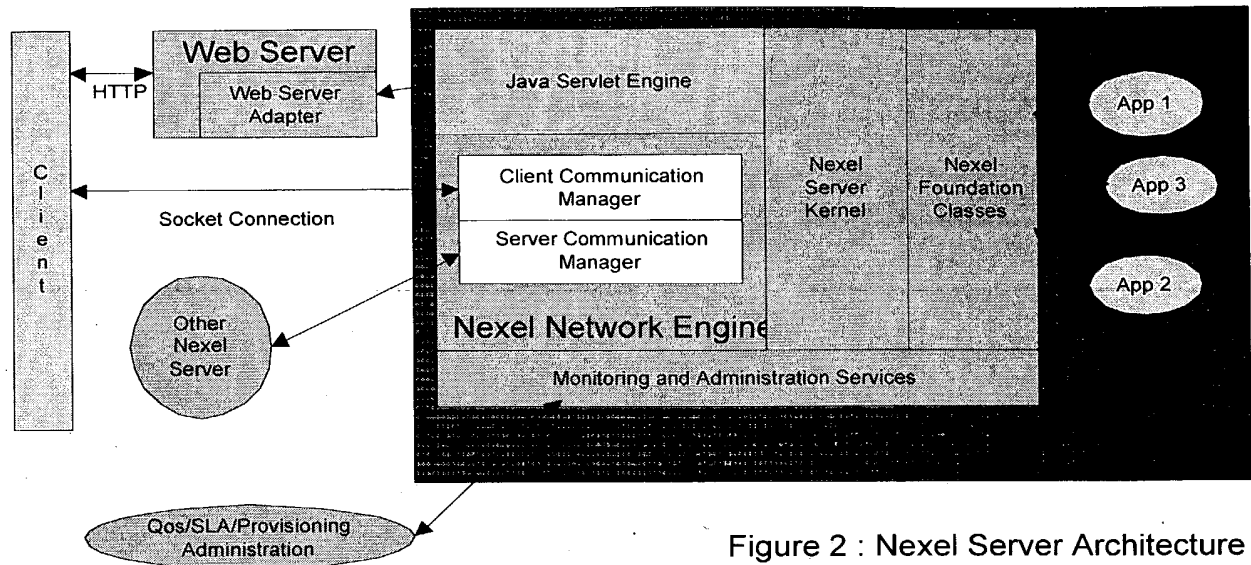


Figure 2 : Nexel Server Architecture

Name	Functionality	Note
------	---------------	------

TABLE 1

Name	Functionality	Note
Web Server Adapter	Communicates with the Java Servlet Engine, such as routing requests to and responses from the Servlet Engine.	Implementation is not needed for phase 1 development. We can use Java Web Server from the Tomcat Project.
Web Server Adapter (132)	Communicates with the Java Servlet Engine, such as routing requests to and responses from the Servlet Engine.	We can also use Java Web Server from the Tomcat Project.
Java Servlet Engine	Provides a standard interface for Nexel Server Kernel to interface with web servers. This standard interface is Java Servlet API. Provides thread management capability. Including: 1). Maintaining a thread pool; 2). Assigning or creating a thread to	The Servlet Engine can be any Servlet Engine, such as JRun and WebLogic. Implementation is not needed for phase 1 development. We can use Tomcat/JWSdk.

	each request, and invokes the "service" method in Nexel Server Servlet from this thread.	
Java Servlet Engine (134)	<p><u>Provides a standard interface for Nexel Server Kernel to interface with web servers. This standard interface is Java Servlet API.</u></p> <p><u>Provides thread management capability.</u></p> <p><u>Including: 1). Maintaining a thread pool; 2). Assigning or creating a thread to each request, and invokes the "service" method in Nexel Server Servlet from this thread.</u></p>	<p><u>The Servlet Engine can be any Servlet Engine, such as Jrun and WebLogic.</u></p> <p><u>We can also use Tomcat/JWS DK.</u></p>
Nexel Server Kernel	<p>Instantiates other Nexel components such as AppManager, ConnectionManager, EventManager and so on. Keeps these components within memory all the time;</p> <p>Launches applications;</p> <p>If necessary, launches another Nexel Server.</p> <p>Event (comes from Servlet requests) dispatching.</p> <p>Generates and sends application user interface description into Servlet responses.</p> <p>If necessary, re-do application user interface layout according the client display characteristics.</p>	<p>Written entirely in Java.</p> <p>Need to be implemented.</p>

<u>Nexel</u> <u>Server</u> <u>Kernel</u> <u>(136)</u>	<u>Instantiates other Nexel</u> <u>components such as</u> <u>AppManager.</u> <u>ConnectionManager.</u> <u>EventManager and so</u> <u>on. Keeps these</u> <u>components within</u> <u>memory all the time;</u> <u>Launches applications;</u> <u>If necessary, launches</u> <u>another Nexel Server.</u> <u>Event (comes from</u> <u>Servlet requests)</u> <u>dispatching.</u> <u>Generates and sends</u> <u>application user interface</u> <u>description into Servlet</u> <u>responses.</u> <u>If necessary, re-do</u> <u>application user interface</u> <u>layout according the</u> <u>client display</u> <u>characteristics.</u>	<u>Written</u> <u>entirely in</u> <u>Java.</u>
<u>Nexel</u> <u>Network</u> <u>Engine</u>	<u>Accepts and creates synchronous</u> <u>socket connections to clients</u> <u>directly, by passing the Servlet</u> <u>Engine and Web Server;</u> <u>Thread management: Maintain a</u> <u>separate thread pool. Whenever a</u> <u>new request is received, creates a</u> <u>ServletRequest and a</u> <u>ServletResponse object, calls the</u> <u>Servlet "service" method within a</u> <u>separate thread; returns this</u> <u>thread back to thread pool after</u>	<u>Written entirely in</u> <u>Java.</u> <u>Needs to be</u> <u>implemented. Open</u> <u>source code from</u> <u>Tomcat project will be</u> <u>helpful.</u>

	the "service" method returns; Communicates with other Nexel Server instances within other Java Virtual Machines through socket connections. The goal of such communication is to route requests/responses to different JVMs such that one JVM won't be overloaded.	
<u>Nexel Network Engine (138)</u>	<u>Accepts and creates synchronous socket connections to clients directly, by passing the Servlet Engine and Web Server; Thread management: Maintain a separate thread pool. Whenever a new request is received, creates a ServletRequest and a ServletResponse object, calls the Servlet "service" method within a separate thread; returns this thread back to thread pool after the "service" method returns; Communicates with other Nexel Server instances within other Java Virtual Machines through socket connections. The goal of such communication is</u>	<u>Written entirely in Java. Open source code from Tomcat project can also be used.</u>

	to route requests/responses to different JVMs such that one JVM won't be overloaded.	
Monitoring and Administration Service	<p>The Monitoring and Administration Service is responsible for keeping track of information about an Application Instance. Following information needs to be tracked:</p> <ul style="list-style-type: none"> a. Application Name. b. Application Instance Identifier. c. User accessing the application. d. Client machine information: <ul style="list-style-type: none"> I. Machine IP II. Device Type. III. Connection Speed. e. Application start time. f. Application last access time. g. Memory usage. h. Active Screen Name 	
Monitoring and Administration Service (140)	<p>The Monitoring and Administration Service is responsible for keeping track of information about an Application Instance. Following information needs to be tracked.</p> <ul style="list-style-type: none"> a. Application Name. b. Application Instance 	

	<u>Identifier.</u> <u>c. User accessing</u> <u>the application.</u> <u>d. Client machine</u> <u>information.</u> <u>I. Machine IP</u> <u>II. Device</u> <u>Type.</u> <u>III. Connection</u> <u>Speed.</u> <u>e. Application start</u> <u>time.</u> <u>f. Application last</u> <u>access time.</u> <u>g. Memory usage.</u> <u>h. Active Screen</u> <u>Name</u>	
--	---	--

Engine, a simple Java Web Server, and adapters from IIS, Apache, Netscape Web Server, will be shipped as part of our product.

Nexel Server Kernel 136 is composed of three major parts:

4.2.1 Nexel Core Classes

- 5 Nexel(com.nexaweb.server.Nexel),
AppManager,EventManager,JVMManager,UIManager,
ConnectionManager,PerformanceManager.

- 10 Class com.nexaweb.server.Nexel 202: This class is the main class. This class extends
javax.servlet.GenericServlet. As a result, it should run with any Java Servlet Engine.
Figure 3 gives an overview of the functionalities of this class.

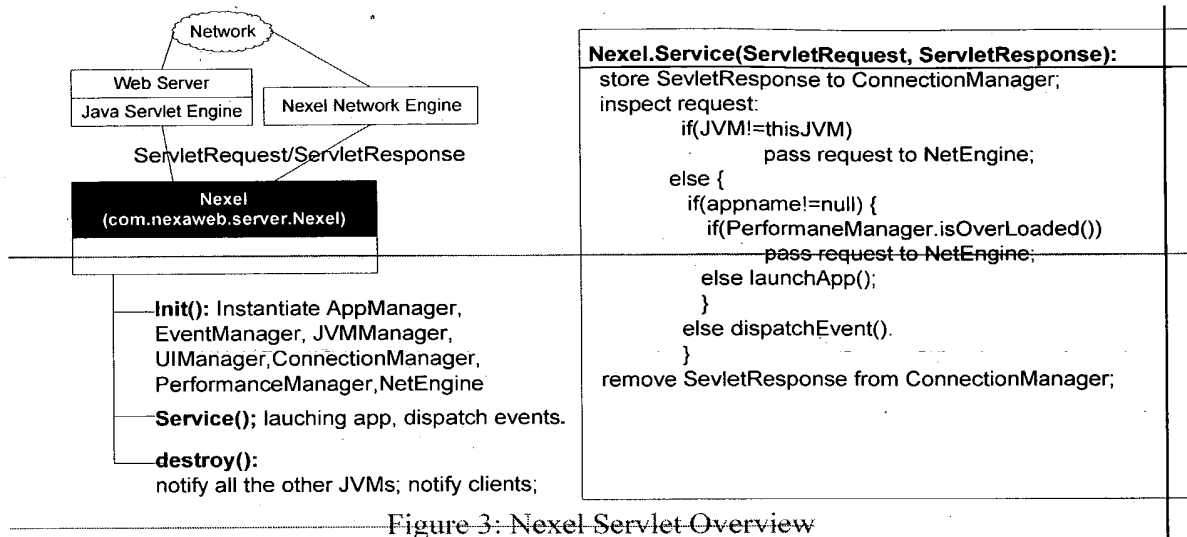


Figure 3: Nexel Servlet Overview

- Class com.nexaweb.server.AppManager: This class manages all the applications within this Java Virtual Machine. Each application is represented as an instance of Class com.nexaweb.server.Application. AppManager maintains a table of applications that can be searched by application ID. It also provides methods for getting the current application.
- Class com.nexaweb.server.Application: This class maintains the persistent data between different threads within one application. It maintains the application ID, a Hashtable of components(Objects) that each component can be located by a unique component ID, a Hashtable of event listeners that each listener can be located by a component ID and an event ID. EventManager and other objects will use these IDs to get access to their interested objects.

The Application class should also maintain an event queue for processing events coming from different threads. See 4.3 Thread-based computing for further information.

- Class com.nexaweb.server.EventManager: this class maintains a map between event ID and the actual event type. It also does the actual event dispatching: for a ServletRequest,

it retrieves appid (application ID). Then it finds the application instance from AppManager by using the appid. It also retrieves the event ID and component ID from the ServletRequest. As last, it constructs corresponding Java Abstract Window Toolkit (AWT) or Swing events and sends the event to the Application object for processing.

- 5 Until the Application object finished the event processing, EventManager should make the current thread idle instead of returning to its caller. After the Application object finished processing this event, the current thread will be destroyed (or recycled).

Class `com.nexaweb.server.JVMManager`: this class maintains a Hashtable of running
10 Java Virtual Machines (each of which has Nexel Server running). The key to the Hashtable is the IP address of the host where the JVM is running and the main port number that the Nexel Network Engine is assigned to listen to during its startup. During event processing, Nexel servlet first check whether this event is for the current JVM. Otherwise it will simply routes the ServletRequest to the corresponding JVM. This class
15 provides methods for setting/getting JVM id and methods for routing requests to different JVMs.

Class `com.nexaweb.server.ConnectionManager`: This class maintains a Hashtable of
ServletResponse instances. Each instances corresponds to a socket communication
20 channel. Each instance can be uniquely located by the name of the thread that this instance belongs to. This class is necessary because the ServletResponse instance cannot be passed to event handlers and UI Objects directly. Event handlers and UI Objects should use this class to find the socket to send information to clients.

25 Class `com.nexaweb.server.UIManager`: This class is a replacement of `javax.swing.UIManager`. It loads a UI class corresponding to the client platform for painting.

Class `com.nexaweb.server.PerformanceManager`: This class provides methods for
30 gauging the performance level of the current JVM. Methods in the class will be used for developing monitoring applications and making load balancing decisions. For example, if

there performance level of the current JVM is below a certain threshold, the Nexel servlet may simply creates a new JVM for handling additional user applications.

InitialOne implementation of the above classes are shown in Appendix 1. is shown in Tables 46-48.

5

4.2.2 Nexel Java Foundation Classes

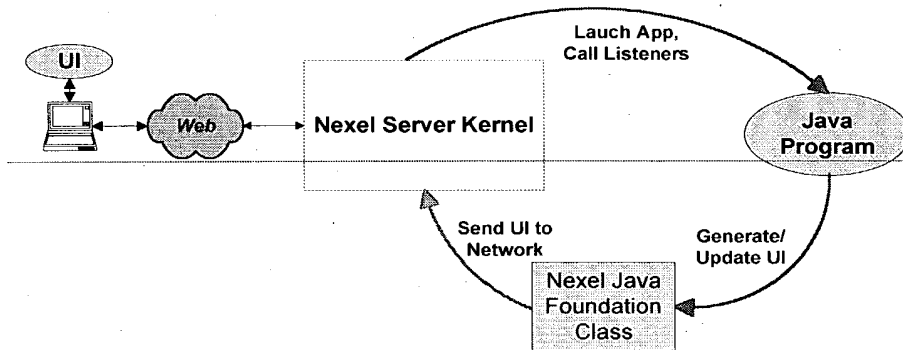


Figure 4: Nexel Java Foundation Class

10

Java Foundation Class, or JFC, is a loose collection of standard Java APIs for client-side graphics; graphical user interfaces (GUIs), and related programming tasks. They are the foundation classes in the sense that most client-side Java applications are built upon these APIs. More in detail, JFC is composed of four includes the following parts:

1. AWT. The Abstract Window Toolkit initially introduced. AWT is the foundation of Swing.
2. Swing. Swing is an advanced GUI toolkit written in pure Java.
3. Java 2D: this is a collection of classes offer two-dimensional graphics.
4. Printing. Both AWT and Swing offer support for printing text and graphics.
5. Data transfer, this includes cut & paste and drag drop.

20

Nexel is an enterprise application presentation platform that delivers an application user interface to any device over any network connection, without modifying existing Java applications. Java applications typically build their UI on top of Java Foundation Class.

- Nexel modifies the behavior of JFC by replacing part of the implementation with its own implementation. This modified JFC is called “Nexel Java Foundation Class” 160, shown in FIG. 4. Instead of painting UI on local computer screen, Nexel Java Foundation Class directly sends UI commands and directives 162 to the client device 112 over a network connection 130. However, Nexel Foundation Class keeps exact the same interface as Java Foundation Class. Namely, for any method in any class, they have the same signature. The only difference is the implementation of these methods. As a result, existing Java applications built using JFC can run as it is without modification. Developers can develop applications to run on top of Nexel without the need of learning anything new.
- 10 In the initial implementation, Nexel aims to support Swing and/or Java 2D based Java applications, but not AWT-based GUI applications. Nexel also plans to support printing and data transfer.

4.2.3 Technical Approach to implement NJFC

- 15 JFC is implemented by following a Model-View-Controller architecture. Each JFC component has a Model, a view and a controller. For example, for the JButton component, its model is AbstractButtonModel, its controller is JButton and its view is JButtonUI (depends on the look & Feel). Sometimes the controller is combined with the model into one class.
- 20 To successfully replace JFC, for each component, we need to re-implement:
1. The component itself (the controller). For example, JButton needs to be re-implemented to achieve three things: a) all event listeners will be added to the Application instance instead of storing within the JButton class itself. Nexel Client Kernel will only send message to Nexel Server if and only if there is an event listener registered for a particular kind of event. So when an event listener is added, a directive indicating that Nexel Server is interested in such event should be sent to the client. 2). Actions for updating/painting the JButton should be sent to UIManager and further sent to the client device; c). When this JButton is created, assign it a unique component ID and store it in the Application Instance.
- 25
- 30

2. It's UI Class (Viewer). For each client platform, a view class needs to be implemented. This view class will be loaded if the client platform matches. This dynamic loading is achieved by UIManager. UIManager needs to check the client platform and loads the view class for that client platform. For example, for JButton, we need to implement WinJButtonUI, MacJButtonUI, XwinJButtonUI, and WinCEJButtonUI.

The functionalities of this UI class are: a). to generate and send UI description of this component to Nexel Client Kernel via the ServletResponse object. The UI description is platform-dependent. It is a protocol that needs to be agreed between the Client Kernel and the UI class on the server side. b). to update UI on the client machine according server-side instructions. The update is accomplished by generating and sending a message describing what and how to update the component UI.

For painting, whether the target output is the client machine display device or a remote printer needs to be taken into design consideration.

For more information, see Design Specification "Nexel Client Kernel".

3. The existing JFC Model classes are used in Nexel Java Foundation Class without modification.

A list of Swing components 300 that we need to replace is shown in Figure 5.

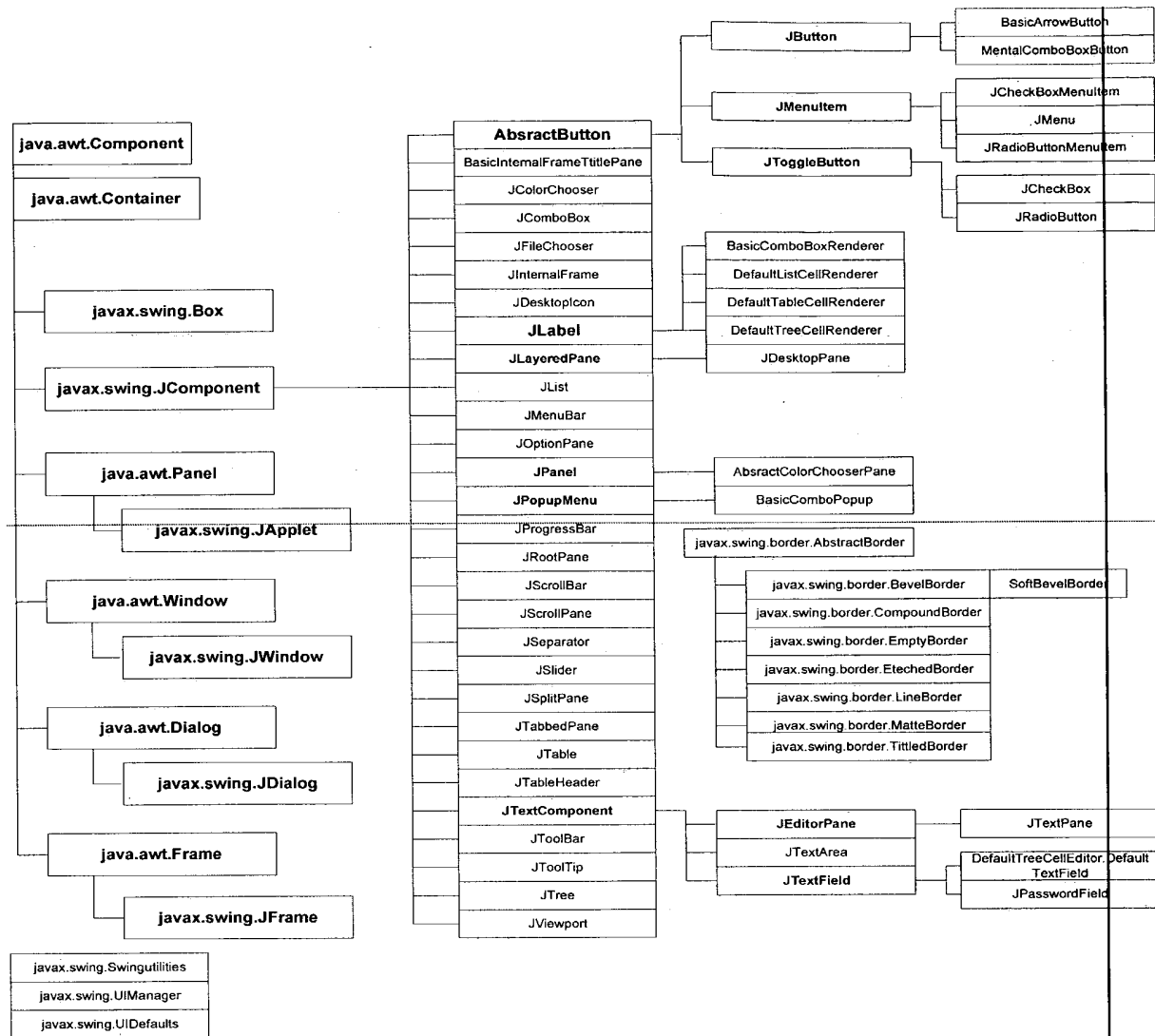


Figure 5: Swing Components that need to be replaced. An UI class is needed for each component for each client platform.

5

TABLE 2

New Components	
Component	Description

JRadioList	List Box with Radio buttons for each item
JCheckList	List Box with Check buttons for each item
JDate	Date editing Control
JCalendar	Calendar Control
JStatusBar	Status Bar Control
JValidation	Runs validation on control values on the client side
New Dialogs	
JPrintDialog	Printer selection dialog

4.2.4 Nexel Layout Manager

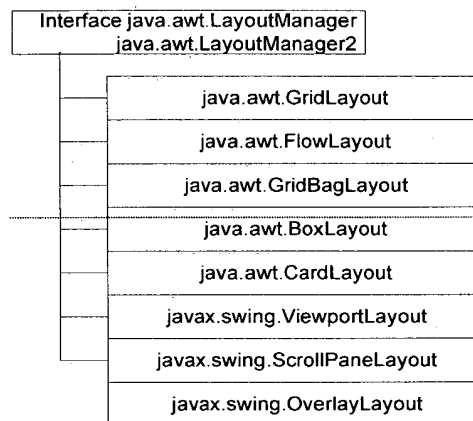


Figure 6: JFC/AWT Layout Managers that needs to be replaced

5

Figure 6 shows a list of layout managers 400 that need to be replaced. Nexel Layout Managers should be adaptive to client display characteristics.

A deployment tool/design tool for laying out GUI components for different client devices will be also provided. See “6 Nexel Client Display Engine” for more information.

10

4.2.5 Nexel Network Engine

Nexel Network Engine is packaged into one package com.nexaweb.server.net. This package is responsible for direct synchronous communication with clients and other Java Virtual Machines that Nexel is running.

15

Class `com.nexaweb.server.net.NetEngine`:

This is one of the two entrances of the Nexel Server (The other one is via a Java Servlet Engine). This class creates instances of `NexelServerSocket`, each of which listens to a pre-determined port. It instantiates `com.nexaweb.server.Nexel` if it has not been

5 instantiated;

~~Within~~within a single JVM, only one instance of Main is allowed. If this JVM is started by another Nexel from a different JVM, this class's main method will be called with arguments that gives the IP address and port number corresponding to the calling JVM. Upon this JVM is started, a message should be sent to the calling JVM notifying "ready" status.

10

Class `com.nexaweb.server.net.NexelServerSocket`:

———This class extends `java.net.ServerSocket`. The instance of this class listens to a specific port. Whenever it accepts a connection, it creates a new thread and lets the new thread (an instance of `SocketHanlder`) handles that connection. Then it returns immediately to listening to that port.

15

Class `com.nexaweb.server.net.SocketHandler`:

———This class extends `java.lang.Thread`. The instance of this class reads/writes to a socket connection. When reading, it formats the information into an instance of `NexelServletRequest` and wraps this socket connection into an instance of `NexelServletResponse`. Passes both objects to the "service()" method in `com.nexaweb.server.Nexel Servlet`.

20

25 Class `com.nexaweb.server.net.NexelServletRequest`:

———This class extends `javax.servlet.ServletRequest`. It basically wraps information coming from a socket connection into a `ServletRequest` Object so that they can be accessed using Java Servlet API.

30 Class `com.nexaweb.server.net.NexelServletResponse`:

———This class extends javax.servlet.ServletResponse. It basically wraps a socket connection into a ServletResponse Object so that they can be accessed using Java Servlet API.

5 **4.2.6 Additional Classes**

The following classes ~~are planned to be replaced:~~
will also be replaced:

1. Graphics 2D:

- 10 AWT provides basic graphics functionalities through instances of java.awt.Graphics. JFC further extends this by extending java.awt.Graphics into java.awt.Graphics2D. These two interfaces are responsible for all the drawing actions.

We need to provide Nexel implementation of java.awt.Graphics and java.awt.Graphics2D interfaces. Whenever method getGraphics() is called(This method is from java.awt.Component, java.awt.Image, java.awt.PrintJob and javax.swing.Jcomponent), we should return an instance of Nexel implementation of java.awt.Graphics2D. This instance should route all the drawing activities to Nexel Client Kernel.

- 20 Both AWT and Graphics2D provide additional classes such as java.awt.Rectangle and the entire java.awt.geom package for manipulating 2D graphics objects (such as affine transformation, setting/getting attributes and so on). These classes do not need to be modified.

2. Printing

AWT providing printing capability by java.awt.PrinterJob. JFC provides printing capability by offering a new package “java.awt.print”. We plan to enable JFC-based printing.

- As a result, java.awt.print.PrinterJob needs to be re-implemented: a). Its static method “getPrinterJob()” needs to be re-written so that a PrinterJob that represents the client machine printer is returned; b). Methods pageDialog() and printDialog() should open dialogs on client machine; c). Method print() should create an instance of java.awt.Graphics2D according to the characteristics of the client machine printer, and pass this instance as an argument for calling “print()” method in the pageable or printable object. Every drawing action happening to the Graphics object should be routed to the client machine for printing.

3. Data Transfer

- Data transfer refers to the ability of an application to transfer selected data I a variety of ways such as Cut&Paste and Drag&Drop. Nexel Platform plans to support both. We plan to support data transfer among different applications on the same client machine. These applications include Nexel-based applications and other native applications. This feature will be enabled by default and its implementation is dependent on the native system.

- Java provides two package called “java.awt.datatransfer” and “java.awt.dnd ”. We need to selectively implement classes in these packages to enable data transfer on remote client machine. Details need to be further studies.

4. Accessibility

- This needs to be further studied. In the short term, we delegate accessibility to the native operating system on the client machine.

5. java.awt.Toolkit.

This class needs to be re-implemented based on Nexel’s distributed presentation model.

The following classes are ~~planned to~~will be added in addition to those offered by Java API:

1. Validation classes.
- 5 2. Other support classes.

4.2.7 Monitoring and Administration UI

The Monitoring and Administration UI is responsible for displaying the information tracked by the Monitoring and Administration Service. This UI will be developed using Java and JFC. Nexel server will be used to web enable this application. The application will contain main screen, which will have a table view displaying all the information. There will be a filter window to filter the entries in the table. The filter should be available for all the information types. This application should also be able to display the active screen in the application.

4.2.8 Thread-based Computing

Nexel is based on a thread-based computing model that every request for the client machine is handled in its own thread. Once the processing is finished, that thread will simply be destroyed or recycled for other usage. Nexel does not maintain a constant process or even a constant thread for an application during its entire session. This is the computing model used in Java Servlet and has proven to be very scaleable.

However, for efficiency reasons, Swing components are not designed to be thread safe. This means that Swing components should be manipulated by a single thread at a time. Nexel Platform needs to pay special attention to this issue during design time:

1. Client-side initiated threads. When a client request is received, either the Java Servlet Engine or Nexel Network Engine will allocate a thread for handling this request. Once the request is processed, this thread will be freed. However, for different client requests,

if they belong to the same application, they need to be processed sequentially though they are in different threads. An event queue for each application instance needs to be maintained.

- 5 2. Server-side initiated thread. There are times that developers need to update UI in response to some kind of external event, such as a response from a server that arrives in a separate thread. To accommodate these situations, Swing provides two utility methods that allow developers to ask the event queue to run arbitrary code. These methods are SwingUtilities.invokeLater() and SwingUtilities.invokeLaterAndWait(). Nexel platform needs
10 to re-implement these methods since Nexel does not use the standard JVM event queue thread.

4.2.9 Performance.

15 Nexel has to provide application delivery at acceptable performance. The time is spent in following actions while running an application under Nexel environment.

- 20 a. Sending instructions from client to server to launch an application. This time should be same as other web application
- b. Launch Application Instance. Attempt should be made to optimize this time, as it may be significant. One way to reduce this is to launch each application as a different thread rather than a new process with new JVM. This poses certain restriction on having static variables.
- 25 c. Extraction of UI. This time significantly depends on how the Component Framework is implemented. It should be very optimized to reduce the time it needs to record its UI.
- d. Transferring UI to client. This time should be same as other web applications.

30 Even though it is very difficult to define the parameters at this time, the performance and scalability should be very close to HTML applications.

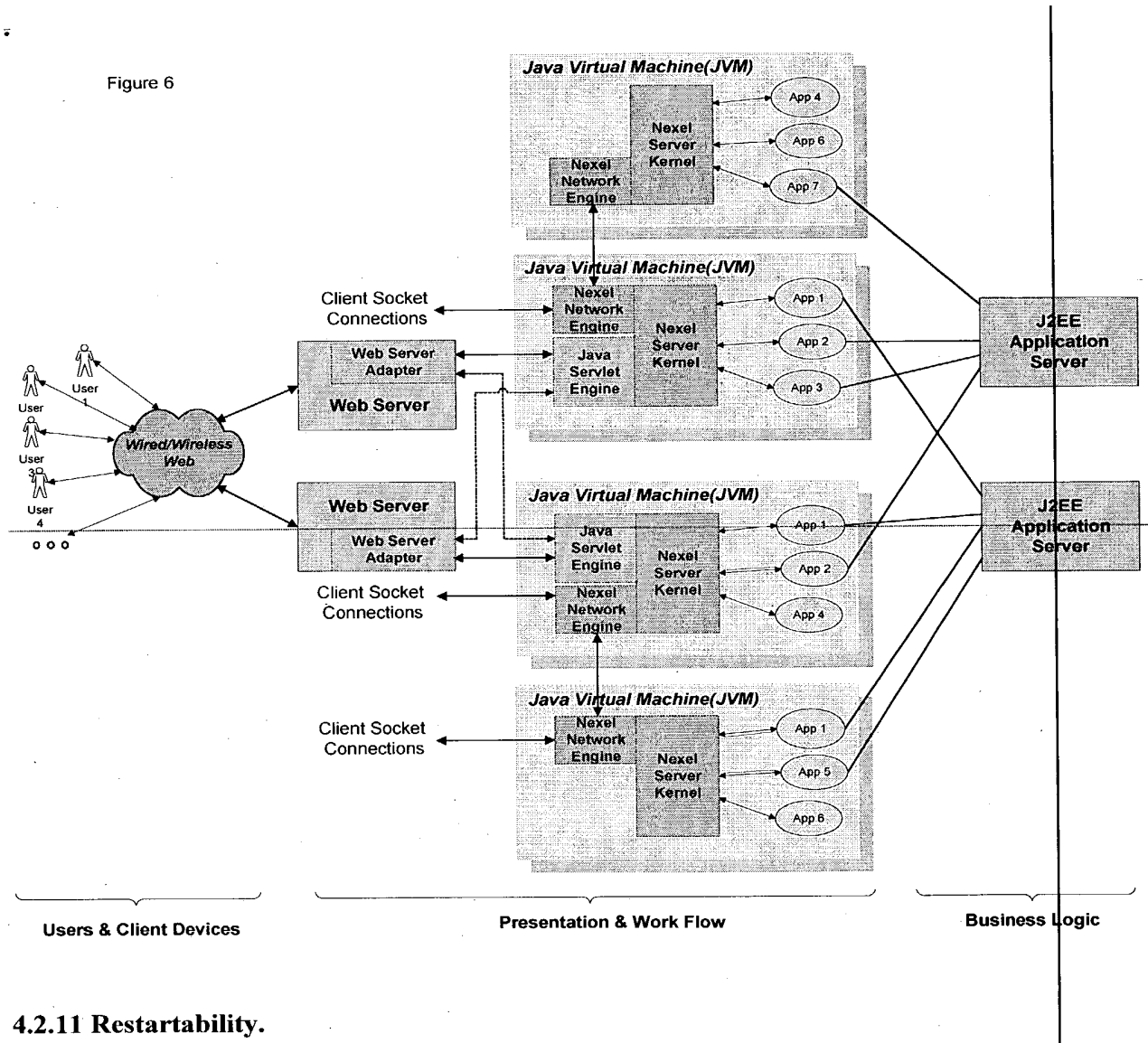
4.2.10 Scalability.

Nexel has to provide highly scalable applications. The server should be capable of running many application instances on server. Every application that is launched consumes certain amount of memory and the design should provide for ways to reduce this consumption. The ways to reduce this consumption are

- a. Serializing the application components to the disk. This frees up memory for new instances to be launched. This at the same time will effect in slower performance.
- b. Allowing application to discard lot of component model memory, which is not needed. For example in a tree control, if the application only concern about the selected item, which could be received from the client, then it can discard the memory for other tree items.
- c. Pooling many JVM or computers into a Server Farm. Nexel Server could run on many machines and it could use different machine to launch many instances.
- d. Adding more web server machine to Web server installation. This feature is found in many Web Servers (e.g. IIS) and could be utilized.

Figure 67 shows an enterprise-scale deployment of Nexel platform and its role in enterprise computing. Nexel Network Engine links different Nexel instances together and forms a high performance-computing environment with unlimited scalability. The diagram has two components

Figure 6



4.2.11 Restartability.

5

The applications run on the server machine should be accessible as long as they are running. If for any reason client breaks connection with the server, the same user which started the application should be able to recreate its UI. Server side application has to maintain information to be able to recreate UI. Some of the changes that user did on

10 client side may not be retractable. The client kernel also reuse some of its cached information in order to recreate the screen.

5. Nexel Client Kernel.

5 Nexel Client Kernel 114 provides functionality to play the application UI on the client machine. It receives the instructions from the NexaWeb UI server for creating and updating the applications UI. It captures all the events and provides default response to events that are not handled by application. Applications could decide to handle events on client side or server side depending on which Kernel will either execute specified code on client side of the application or notify the event on server side of the application. It will
10 also wait for response from server and update the UI when asked. It also provides caching of an application UI on the client side

5.1 Nexel Client Kernel Platforms.

15 Client kernel needs to be implemented on many platforms that play an application UI. For Release 1.0 only In one example, three platforms are targeted, shown in table 3.

TABLE 3

20

Platform	Technology	Language for development
Internet Explorer	Active-X component	C++
<u>Internet Explorer</u>	<u>Active -X component</u>	<u>C++</u>
DHTML/JS code. Lot of this code exists today. Should be finished and tried out		DHTML/JS
	Applet	Java
Netscape Browser	Plugin	C++
Windows CE	Proprietary player needs to develop.	C++

<u>Windows CE</u>	<u>Proprietary player</u>	<u>C++</u>
-------------------	---------------------------	------------

- 5 The same code base is expected for Active-X component, Netscape Plugin and Windows CE player. These players will be developed using C++ language and so the code base should be same. On windows platform ATL Windows classes should be used to implement

5.2 Nexel Client Kernel Architecture.

- 10 The core functionality of the Client Kernel is to provide a rich Component Framework to create rich user interface. It creates instances of these components on the user's display and monitor events. The architecture for Client Kernel is shown in FIG. 8 and is also summarized in Table 4 below.

15

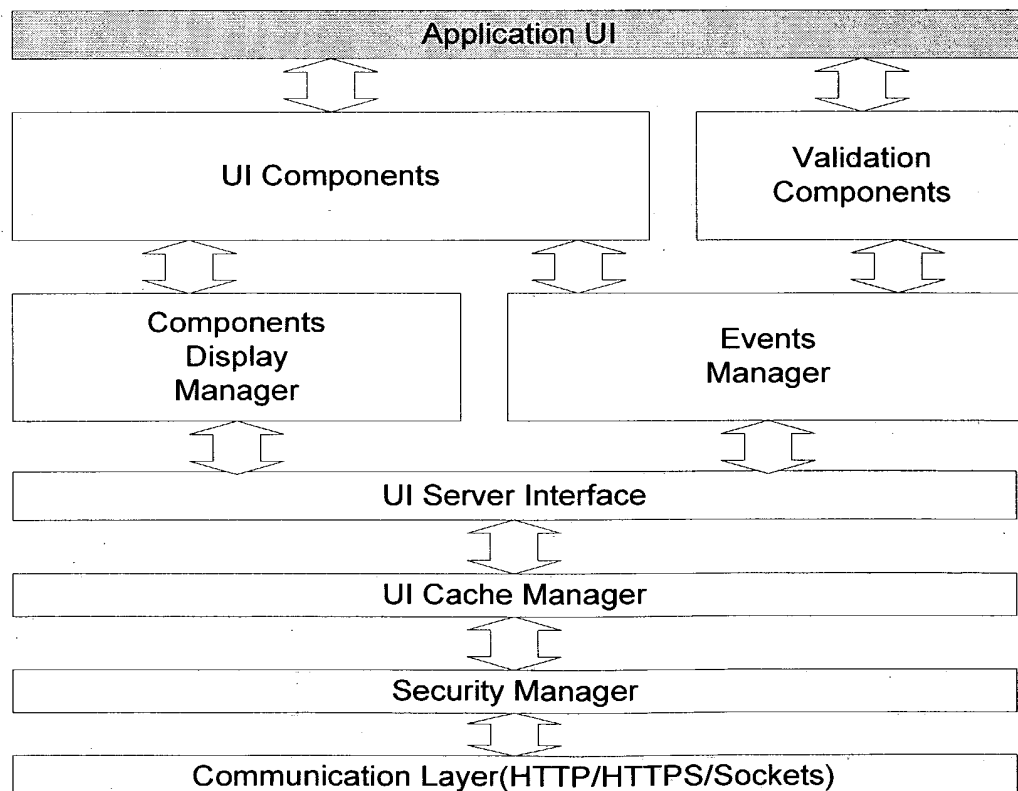


Figure 1 : Nexaweb Client Kernel Architecture

UI Components	This is a widget set that is implemented in the Kernel. It includes various widgets like button, window, tree etc.
---------------	--

TABLE 4

Validation Components	These are components provides functionality for simple validation of data. These components are invoked on specified events and they execute specified validation rules
-----------------------	---

<u>UI Components</u> (502)	<u>This is a widget set that is implemented in the Kernel. It includes various widgets like button, window, tree etc.</u>
<u>Validation Components</u> (504)	<u>These are components provides functionality for simple validation of data. These components are invoked on specified events and they execute specified validation rules</u>

Components Display/Update Manager	This takes care of creating UI Components and updating them whenever necessary
-----------------------------------	--

<u>Components Display/Update Manager</u> (506)	<u>This takes care of creating UI Components and updating them whenever necessary</u>
---	---

Cache Manager	This manages Client Kernel Caching as well as application UI caching on the client side. On Windows CE Platforms it should bring up an application even though it is not connected to the server.
---------------	---

<u>Cache Manager</u> (512)	<u>This manages Client Kernel Caching as well as application UI caching on the client side. On Windows CE Platforms it should bring up an application even though it is not connected to the server.</u>
-------------------------------	--

Event Manager	Event manager handles all the events
---------------	--------------------------------------

<u>Event Manager</u> (508)	<u>Event manager handles all the events</u>
-------------------------------	---

UI Server Interface	This module convert the packets received from the server into various UI and event monitoring commands.
---------------------	---

Security Manager	Handles security for communicating with the server. This determines the protocol and encryption that will be used.
UI Server Interface (510)	This module convert the packets received from the server into various UI and event monitoring commands.
Security Manager (514)	Handles security for communicating with the server. This determines the protocol and encryption that will be used.
Communication Layer	This layer provides services for communicating with the server.
Communication Layer (516)	This layer provides services for communicating with the server.

5.3 Security Manager.

5

The Security Manager 514 handles security for the communication records going to and from server to client. Standard security is used when a Web Server is involved. The HTTPS protocol is used to handle security. For socket-based communication, our own security mechanism will be implemented. Evaluate third party products

10 to do this.

5.4 UI Cache Manager.

The Cache Manager 512 has two purposes:

- 15
- To cache the Client Kernel itself. The Kernel detects that there is a newer version of the Kernel available on the server and it downloads and runs that Client Kernel. For browser-based implementations, its own caching mechanism can be used. For non-browser based implementations such as Windows CE, it will have to be implemented.

- To cache the application UI. An application name and its version will be used for caching. The code detects application versions on the server and client and if they match, it will not download its UI from the server. The server can disable the caching. The different windows that are opened during different events will also be cached. This feature can be used to play the UI on a client even if there is no connection to the server.

The cache manager uses a cache directory to save files. This directory will be identified by the

Registry key on the windows platform:

10 HKEY_CURRENT_USER\Software\NexaWeb\Nexel\Cache Directory

Environment variable on other platforms:

NEXEL_CACHEDIR

The directory structure used for caching should look like this:

```

15 + HOME
      + Application Name
          + Version
              - File1
              - File 2
20              - EventWindow.map

```

The directory contains a file EventWindow.map that keeps track of an event and the window that was opened due to the event.

25 6_ UI Record Format.

The server sends the UI information to the client using this record format. Records are written in XML and packaged in text and transmitted over http protocol. This format may need tuning and can be changed to a compact binary format. The XML format is implemented, at least for testing purposes. The following principles are implemented:

- Each application instance has an assigned identification number.

- Each window is identified with an identifier from the server. This identifier is used in later communications for event handling and property changes.
 - Every control implements properties, methods and events of its base class. Window class is the base class for all the controls.
- 5
- Every control is identified by a class name. This name is identified by the Widget name in UI Components table.
 - The control's properties are described after identifying a window.
 - The events that are to be handled are described later on.
- 10 Applications properties are listed in following table 5.

Application

15

TABLE 5

Application			
Properties			
Name	Description	Structure/Possible Values	XML Examples
Name	Description	Structure/Possible Values	XML Examples
Id	Identifier		id=123

	for App instance		
Name	Application Name		name="MyApp"
Version	Application Version		version="1.0"
Cache	Use Caching	true,false	cache=true

A typical record may look like this:

```

5      <nexawebapp id=123 name="myapp" version =1.0 cache=true>
      <window id=1>
          <bkcolor>yellow</bkcolor>
          <text>My First Application</text>
          <toolbar id=2>
              <button id=3 command=100>
10                 <text>New</text>
                  <image>images/new.gif</image>
              </button>
              <button id=4 command=101>
15                 <text>Exit</text>
                  <image>images/new.gif</image>
              </button>
          </toolbar>
          <oncommand id=100></oncommand>
          <oncommand id=101></oncommand>
20      </window>
      </nexawebapp>

```

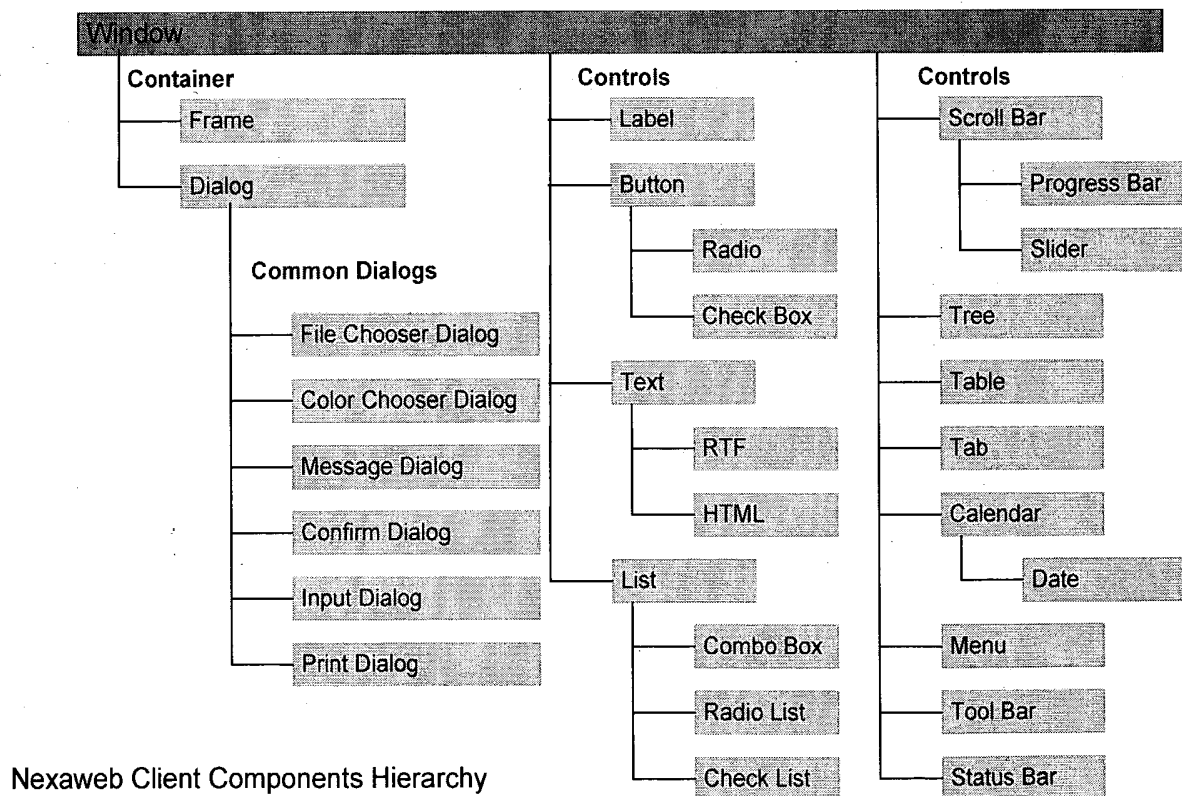
6.1 Event Record Format.

The Event Record Format specifies how the event records are passed back to the server. Events are passed back with application ID and a Window ID. It also collects the properties that are specified to be collected.

A typical event record may look like this:

```
<nexawebapp id=123>
  <window id=1>
    <oncommand id=100></oncommand>
  </window>
</nexawebapp>
```

7. Client Component Hierarchy.



The client component hierarchy 600 is shown in FIG. 9 and the following table 6 displays the UI Components that are supported. Each Control supports its properties, methods and events. The following section describes each control that is implemented by the Client Kernel.

TABLE 6

Components		
Widget Name	Window Equivalent	Description
Component	Window	Basic Window container
Dialog	Dialog	Dialog container
Frame	FrameWnd	MDI Frame container
Label	Static	Caption or label control
Push Button	Button	Push Button Control
Radio Button	Button	Radio Button Control
Check Button	Button	Check Button Control
Text	Edit	Edit/Text Control
RTF	RichEdit	Rich Text Edit
HTML	DHTMLEdit	HTML control
RTF	RichEdit	Rich Text Edit

<u>HTML</u>	<u>DHTMLEdit</u>	<u>HTML control</u>
List	ListBox	List Control
Combo Box	ComboBox	Combo Box control
Radion List		List Box with Radio buttons for each item
Check List		List Box with Check buttons for each item
Progress Bar	ProgressCtrl	Progress Bar Control
Slider	SliderCtrl	Slider Control
Tree	TreeCtrl	Tree Control
Table	ListCtrl	Table Control
Tabbed Dialog	Tab	Tabbed Dialog
Date	DateTimePicker	Date editing Control
Scroll Bar	ScrollBar	Scroll Bar Control
Calendar	Calendar	Calendar Control
<u>Menu</u>	<u>Menu</u>	<u>Menu</u>
<u>Menu</u>	<u>Menu</u>	<u>Menu</u>
Tool Bar	ToolBar	Tool Bar control
Status Bar	StatusBar	Status Bar Control
Dialogs		
Message Dialog	MessageBox	Message dialog with OK and CANCEL button
Confirm Dialog	MessageBox	Confirm Dialog with YES,NO, CANCEL buttons
Input Dialog		Input Dialog with one edit control
File Dialog	FileDialog	File Selection Dialog
Color Dialog	ColorDialog	Color Chooser Dialog
Print Dialog	PrintDialog	Printer selection dialog

8. Nexel Communication Format.

- 5 Nexel provides many ways of communication between Client Kernel and Delivery Server. For each client type there could one implementation. In this release only one format is to be implemented. The communication happens in the UI record format and Event Record format. The following tables 7-46 describe the properties, methods and event that each component could have.

TABLE 7

Component			
Properties			
Name	Description	Structure/Possible Values	XML Examples
location	Window Location	Left, top, width, height Left or top = -1 locate anywhere Width or height = -1 freely size	location="10,10,-1-1"
bkeolor	Background color	Color Value	bkeolor=red

<u>bkcolor</u>	Backgro und color	Color Value	<u>bkcolor=red</u>
<u>textcolor</u>	Foreground color	Color Value	<u>textcolor=green</u>
<u>textcolor</u>	Foregro und color	Color Value	<u>textcolor=green</u>
state	Window state	minimized maximized normal	state=normal
minimizeb ox	Window has minimiz e box	True,false	minimizebox=tru e
maximizeb ox	Window hax maximiz e box	true,false	maximizebox=tru e
cancelbox	Window has cancel box	true, false	cancelbox=true
systemmen u	Window has system menu	true, false	systemmenu=true
visible	window is visible	true,false	visible=true
enabled	window is	true,false	enabled=true

	enabled		
contexthelp	Window has help button	true,false	contexthelp=false
text	Window title/caption/ text		text="My window"
<u>text</u>	Window title/capt ion/text		<u>text</u> ="My window"
border	Window border	none fixed dialog sizeable	border=none
focus	Window has focus	true/false	focus=true
font	Window font	see font description	
icon	Window Icon	value	icon="images/icon.gif"
<u>icon</u>	Window Icon	<u>value</u>	<u>icon</u> ="images/ico n.gif"
cursor	Mouse pointer		cursor="images/hand.gif"
<u>cursor</u>	Mouse pointer		<u>cursor</u> ="images/h and.gif"
helpurl	Help URL		helpurl="help/mywindow.h tm"
<u>helpurl</u>	Help URL		<u>helpurl</u> ="help/my window.htm"
flash	Flash Window	true, false	flash=true

tooltiptext	Tool Tip Text		tooltiptext="Here is my tooltip"
-------------	------------------	--	-------------------------------------

Methods			
Name	Description	Parameters	XML Examples
print	Print the window		<print></print>
center	Center Window	window id	<center id=10></center >
redraw	Redraw Window		<redraw></redr aw>
move	Move Window	left,top,width. height	<move>10,10,- 1</move>
<u>move</u>	<u>Move</u> <u>Window</u>	<u>left,top,width.hei</u> <u>ght</u>	<u><move>10,10,-</u> <u>1,-1</move></u>
show	Show Window		<show></show >
moveto	Move Drawing point		<moveto x=10 y=10> </moveto>
lineto	Draw Line		<pen>...</pen> <lineto x=10 y=10> </lineto>
<u>lineto</u>	<u>Draw Line</u>		<u><pen>...</pen></u> <u><lineto x=10</u> <u>y=10></u> <u></lineto></u>
circle	Draw Circle		<pen>...</pen> <brush>...</br ush> <circle x=10 y=10 radius=20

			fill=true> </circle>
rectangle	Draw Rectangle		<pen>...</pen> <brush>...</brush> <rectangle x=10 y=10 width=20 height=20 fill=true> </rectangle>
polygon	Draw polygon		<pen>...</pen> <brush>...</brush> <polygon points=10,10;20,20;30,30 fill=true></rectangle>
<u>polygo</u> <u>n</u>	<u>Draw</u> <u>polygon</u>		<pen>...</pen> <brush>...</brush> <polygo n points=10,10;20,20;30,30 fill=true> </rectangle>

Events			
Name	Description	Parameters	XML Examples
Onmousedown		button, x,y	<onmousedown button=left x=10 y=20></mousedown>
onmouseup			<onmouseup button=left x=10

			y=20></mouseup>
Onmousemove			<onmousemove x=10 y=20></mousemove>
onclick			<onclick button=left x=10 y=20></onclick>
ondblclick			<ondblclick button=left x=10 y=20></ondblclick>
onactivate	window is activated		<onactivate oldactiveid=122> </onactivate>
ondeactivate	window is deactivated		<ondeactivate newactiveid=133> > </ondeactivate>
Oncontextmenu	context menu is requested		<oncontextmenu></oncontextmenu>
Oncontextmenu	context menu is requested		<oncontextmenu> </oncontextmenu> >
onfocus	window is getting focus		<focus oldfocusid=111> </focus>
onkillfocus	window is losing focus		<onkillfocus newfocusid=111> </onkillfocus>

onhelp	help is invoked		<onhelp></onhelp>
onkeydown	key is down		<onkeydown key="b" shift=true alt=false ctrl=true> </onkeydown>
onkeyup	key is up		<onkeyup key="b" shift=true alt=false ctrl=true> </onkeyup>
onkeypress	key is pressed		<onkeypress key="b" shift=true alt=false ctrl=true> </onkeypress>
onresize	window is resized		<onresize width=10 height=50> </resize>
onmove	window is moved		<onmove left=10 top=20> </onmove>
oncommand	Window receives a command		<oncommand id=200> </oncommand>

TABLE 8

Dialog			
Properties			
Name	Description	Structure/Possible Values	XML Examples
modal	Dialog is modal	true/false	modal=true
Methods			
Name	Description	Parameters	XML Examples
reset	reset the dialog		<reset></reset>
Events			
Name	Description	Parameters	XML Examples
submit	Submit dialog		<submit></submit>

5

TABLE 9

Frame			
Properties			
Name	Description	Structure/Possible Values	XML Examples
splitdir	Split Direction	horz,vert	splitdir=horz
Methods			
Name	Description	Parameters	XML Examples
split	Split the frame	pane, no of panes	<split pane=0,1 panes=2></split>
attach	Attach	window	<attach

h	Window		windowid=100 > </attach>
Events			
Name	Description	Parameters	XML Examples
e			

TABLE 10

Label			
Properties			
Name	Description	Structure/Possible Values	XML Examples
Methods			
Name	Description	Parameters	XML Examples
Events			
Name	Description	Parameters	XML Examples

TABLE 11

Push Button			
Properties			
Name	Description	Structure/Possible Values	XML Examples
Name	Description	Structure/Possible Values	XML Examples
image	Image	image file name	image="images/yes.gif"
overimage	Image to be displayed		overimage="images/no.gif"

	when mouse is moved over		
	<u>overimage</u>	Image to be <u>displayed</u> when mouse is moved over	<u>overimage="images/ no.gif"</u>
<u>downimage</u> <u>e</u>	button image when pushed down		<u>downimage="images/down.gif"</u>
	<u>downimage</u> <u>e</u>	button image when pushed down	<u>downimage="images /down.gif"</u>
<u>selectedimage</u> <u>age</u>	button image when selected		<u>selectedimage="images/selected -gif"</u>
	<u>selectedimage</u> <u>age</u>	button image when selected	<u>selectedimage="ima ges/selected.gif"</u>
<u>imageloc</u>	image location in reference to text	left,top,right,b ottom	<u>imageloc=top</u>
	<u>imageloc</u>	image location in reference to text	<u>imageloc=top</u>
<u>imagealign</u>	image	left,right,cente	<u>imagealign="top,center"</u>

	alignment	r,top,bottom,v center	
imagealign	image alignmen t	left,right,ce nter,top,bot tom,vcenter	imagealign="top,center"

Methods			
Name	Description	Parameters	XML Examples
Events			
Name	Description	Parameters	XML Examples
oncommand	Button is pushed		<oncommand id=202> </oncommand>
	Button is pushed		<oncommand id=202> </oncommand>
Radio Button			

5

10

15

TABLE 12

Radio Button			
Properties			
Name	Description	Structure/Possible Values	XML Examples
selected	radio button is selected	true,false	selected=true
Methods			
Name	Description	Parameters	XML Examples
Events			
Name	Description	Parameters	XML Examples

5

TABLE 13

Check Box			
Properties			
Name	Description	Structure/Possible Values	XML Examples
checked	button is checked	true,false	checked=true
checked	button is checked	true,false	checked=true
Methods			
Name	Description	Parameters	XML Examples

Events			
Name	Description	Parameters	XML Examples

TABLE 14

Group Box			
Properties			
Name	Description	Structure/Possible Values	XML Examples
Methods			
Name	Description	Parameters	XML Examples
Events			
Name	Description	Parameters	XML Examples

5

TABLE 15

Text			
Properties			
Name	Description	Structure/Possible Values	XML Examples
multiline	whether multiline text	true,false	multiline

textlimit	number of characters allowed	number	textlimit=10
editmask	mask to be used for editing	string	editmask="###.##.####"
<u>editmask</u>	<u>mask to be used for editing</u>	<u>string</u>	<u>editmask="###.##.####"</u>
<u>datatype</u>	<u>type of data</u>	<u>string,number, date,currency, amount</u>	<u>datatype=date</u>
<u>datatype</u>	<u>type of data</u>	<u>string,number, date,currency, amount</u>	<u>datatype=date</u>
outformat	output format	string	outformat="\$%x"
align	Alignment	left,center,right	align=right

Methods			
Name	Description	Parameters	XML Examples
clear	Clear the text		<clear></clear>
cut	Cut the selected text to clipboard		<cut></cut>
paste	Paste from clipboard		<paste></paste>
copy	Copy the selected text		<copy></copy>

	to clipboard		
Events			
Name	Description	Parameters	XML Examples
onchange	Text is changed		<onchange text="1234"></onchange>
onchange	Text is changed		<onchange text="1234"></onchange>

5

10

TABLE 16

RTF			
Properties			
Name	Description	Structure/Possible Values	XML Examples
Methods			
Name	Description	Parameters	XML Examples
Events			

Name	Description	Parameters	XML Examples

TABLE 17

HTML			
Properties			
Name	Description	Structure/Possible Values	XML Examples
Methods			
Name	Description	Parameters	XML Examples
Events			
Name	Description	Parameters	XML Examples

5

TABLE 18

List			
Properties			
Name	Description	Structure/Possible Values	XML Examples
listitem	List item description: many	text,image,data	<listitem text="xyz" image="images/checked.gif" data="a value"></listitem>

	records repeated		
<u>listitem</u>	List item description many records repeated	text,image, data	<listitem text="xyz" image="images/checked.gif" data="avalue"></listitem>
selectedtext	Selected item text		selectedtext="xyz"
selecteddata	Selected item data		selecteddata="avalue"
itemcount	Number of items		itemcount=10
multiselect	Multiple selection Allowed	true,false	multiselect=true
sorted	List is sorted	true,false	sorted=false
Methods			
Name	Description	Parameters	XML Examples
additem	Adds a item	listitem	<additem text="xyz" data="abc"></additem>
Events			
Name	Description	Parameters	XML Examples
selchange	Selection Changed		<selchange index=0

			text="xyz" data="abc"> </selchange>
--	--	--	---

TABLE 19

Radio List			
Properties			
Name	Description	Structure/Possible Values	XML Examples
Methods			
Name	Description	Parameters	XML Examples
Events			
Name	Description	Parameters	XML Examples

5

TABLE 20

Check List			
Properties			
Name	Description	Structure/Possible Values	XML Examples
Methods			
Name	Description	Parameters	XML Examples
Events			
Name	Description	Parameters	XML Examples

5

TABLE 21

Scroll Bar			
Properties			
Name	Description	Structure/Possible Values	XML Examples
direction	Progress direction	horz,vert	direction=vert
range	Range	low,high	range=100,200
range	Range	low,high	range=100,200
pos	Position		pos=10
linestep	Step		linestep=10
pagestep	Page Step		pagestep=100
Methods			
Name	Description	Parameters	XML Examples
scroll	Scroll the bar by offset		scroll=-10
setscroll	Set the thumb		setscroll=100
Events			
Name	Description	Parameters	XML Examples
poschange	Position Changed		<poschange newpos=105> </poschange>

<u>poschange</u>	<u>Position</u> <u>Changed</u>		<u><poschange</u> <u>newpos=105></u> <u></poschange></u>
------------------	-----------------------------------	--	--

5

TABLE 22

Progress Bar			
Properties			
Name	Description	Structure/Possible Values	XML Examples
smooth	Smooth Scrolling bar	true,false	smooth=true
step	Step		step=10
Methods			
Name	Description	Parameters	XML Examples
stepit	Advance		<u><stepit numofsteps=10></u> <u></stepit></u>
stepit	Advance		<u><stepit</u> <u>numofsteps=10></u> <u></stepit></u>
Events			
Name	Description	Parameters	XML Examples

TABLE 23

Slider			
Properties			
Name	Description	Structure/Possible Values	XML Examples
Methods			
Name	Description	Parameters	XML Examples
Events			
Name	Description	Parameters	XML Examples

Calendar

TABLE 24

Calendar			
Properties			
Name	Description	Structure/Possible Values	XML Examples
daterange	Range in Calendar Control	from, to	<daterange from="1/1/99" to="2/2/00"></daterange>
date	Selected date		date="1/1/00"
Methods			
Name	Description	Parameters	XML Examples
Events			
Name	Description	Parameters	XML Examples
selchange	Date		<selchange

	selection changed		newdate="1/1/00"> </selchange>
--	----------------------	--	-----------------------------------

5

10

TABLE 25

Date			
Properties			
Name	Description	Structure/Possible Values	XML Examples
Name	Description	Structure/Possible Values	XML Examples
inputmask	Input Mask		inputmask="mm/dd/yy"
outformat	Output Format		outformat="dd-mmm-yyyy"
outformat	Output Format		outformat="dd-mmm-yyyy"
Methods			
Name	Description	Parameters	XML Examples

Events			
Name	Description	Parameters	XML Examples
e			

TABLE 26

Tree			
Properties			
Name	Description	Structure/Possible Values	XML Examples
Name	Description	Structure/Possible Values	XML Examples
treeitem	Tree Item Structure		<pre><treeitem parent=0 id=1 text="abcd" image="open.gif" openimage="close.gif" selectedimage="selected.gif" children=1 data="ext111" state=expanded></treeitem></pre>
<u>treeitem</u>	<u>Tree Item Structure</u>		<pre><treeitem parent=0 id=1 text="abcd" image="open.gif" openimage="close.gif" selectedimage="selected.gif" children=1 data="ext111" state=expanded> </treeitem></pre>
selecteditem	Selected Tree Item		<pre><selecteditem id="1" text="abcd" data="ext111"> </selecteditem></pre>
rightclickitem	Item where right clicked		<pre><rightclickitem id="1" text="abcd" data="ext111"> </rightclickitem></pre>

	clicked		
Methods			
Name	Description	Parameters	XML Examples
insertitem	Insert Item		<code><insertitem></code> <code><treeitem>...</treeitem></code> <code></insertitem></code>
<u>insertitem</u>	<u>Insert</u> <u>Item</u>		<code><insertitem></code> <code><treeitem>...</tr</code> <code>eeitem></code> <code></insertitem></code>
deleteitem	Delete Item		<code><deleteitem></code> <code><treeitem>...</treeitem></code> <code></deleteitem></code>
<u>deleteitem</u>	<u>Delete</u> <u>Item</u>		<code><deleteitem></code> <code><treeitem>...</tr</code> <code>eeitem></code> <code></deleteitem></code>
deleteall	Delete all items		<code><deleteall></code> <code></deleteall></code>
expanditem	Expand Item		<code><expanditem</code> <code>id=1></code> <code></expanditem></code>
selectitem	Select Item		<code><selectitem</code> <code>id=1></code> <code></selectitem></code>
sortchildren	Sort Children		<code><sortchildren</code> <code>id=1></sortchildr</code> <code>en></code>
ensurevisible	Ensure Visible		<code><ensurevisible</code> <code>id=1></code> <code></ensurevisible></code>

editlable	Edit Label		<editlabel id=1> </editlabel>
setitem	Set Item		<setitem> <treeitem>...</treeitem> </setitem>
<u>setitem</u>	<u>Set Item</u>		<u><setitem></u> <u><treeitem>...</tr</u> <u>eeitem></u> <u></setitem></u>
Events			
Name	Description	Parameters	XML Examples
begindrag	Begin Drag		<begindrag id=1> </begindrag>
enddrag	End Drag		<enddrag id=1 droptarget=5 > </enddrag>
itemexpanded	Item expanded		<itemexpanded id=1 text="xyz" data="as111"> </itemexpanded>
itemexpanding	Item expanding		<itemexpanding id=1 text="xyz" data="as111"> </itemexpanding >
selchanged	Selection Changed		<selechanged id=1 text="xyz" data="as111"> </selechanged>
selchanging	Selection		<selechanging

	Changing		id=1 text="xyz" data="as111"> </selechanging>
--	----------	--	---

5

10

TABLE 27

Table			
Properties			
Name	Description	Structure/Possible Values	XML Examples
<u>Name</u>	<u>Description</u>	<u>Structure/Possible Values</u>	<u>XML Examples</u>
<u>tablecol</u>	Table Header Column	see table column	<tablecol>...</tablecol>
<u>tablecol</u>	Table Header Column	see table column	<tablecol>...</tablecol>
<u>tablerow</u>	Table Row		<tablerow>...</tablerow>
<u>tablecell</u>	Table Cell		<tablecell>...</tablecell>

	Cell		
selectedrows	Selected Row		selectedrows=10,11,12,13
Methods			
Name	Description	Parameters	XML Examples
insertrow	Insert Row		<code><insertrow></code> <code><tablerow>...</tablerow></code> <code></insertrow></code>
<u>insertrow</u>	<u>Insert Row</u>		<code><insertrow></code> <code><tablerow>...</tablerow></code> <code></insertrow></code>
deleterow	Delete Row		<code><deleterow row=1></code> <code></deleterow></code>
deleteall	Delete all Rows		<code><deleteall></code> <code></deleteall></code>
selectrow	Select Row		<code><selectrow row=1></code> <code></selectrow></code>
sort	Sort table		<code><sort column=1></sort></code>
ensurevisible	Ensure Visible		<code><ensurevisible row=1></code> <code></ensurevisible></code>
setrow	Set Row		<code><setrow row=1></code> <code><tablerow>...</tablerow></code> <code></setrow></code>
<u>setrow</u>	<u>Set Row</u>		<code><setrow row=1></code>

			<code><tablerow>...</tablerow></code> <code></setrow></code>
settablecell ↓	Set Table Cell		<code><settablecell row=1 col=5></code> <code><tablecell>...</tablecell></code> <code></settablecell></code>
settablecell	Set Table Cell		<code><settablecell row=1 col=5></code> <code><tablecell>...</tablecell></code> <code></settablecell></code>
insertcol	Insert Column		<code><insertcol after=1></code> <code><tablecol>...</tablecol></code> <code></insertcol></code>
insertcol	Insert Column		<code><insertcol after=1></code> <code><tablecol>...</tablecol></code> <code></insertcol></code>
deletecol	Delete Column		<code><deletecol col=1></code> <code></deletecol></code>
deleteallcol ↓	Delete All Columns		<code><deleteallcol></code> <code></deleteallcol></code>
Events			
Name	Description	Parameters	XML Examples
begindra	Begin		<begindrag

g	Drag		id=1> </begindrag>
enddrag	End Drag		<enddrag id=1 droptarget=5 > </enddrag>
selchange d	Selection Changed		<selechanged id=1 text="xyz" data="as111"> </selechanged>
selchang ing	Selection Changing		<selechanging id=1 text="xyz" data="as111"> </selechanging>

TABLE 28

Tabbed Dialog			
Properties			
Name	Description	Structure/Possible Values	XML Examples
Name	Description	Structure/Possible Values	XML Examples
tab	Tab structure	All button attributes and view attribute	<tab id=1 image="mytab.gif" text="abcd" view="abcd"> </tab>
selectedtab	Selected Tab		<selectedtab> <tab>...</tab> </selectedtab>

Methods			
Name	Description	Parameters	XML Examples
inserttab b	Insert Tab		<code><inserttab></code> <code><tab>...</tab></code> <code></inserttab></code>
deletetab b	Delete Tab		<code><deletetab></code> <code><tab>...</tab></code> <code></deletetab></code>
deleteall l	Delete all tabs		<code><deleteall></code> <code></deleteall></code>
selecttab b	Select Tab		<code><selecttab id=1></code> <code></selecttab></code>
settab	Set tab		<code><settab></code> <code><tab>...</tab></code> <code></settab></code>
Events			
Name	Description	Parameters	XML Examples
selchanged d	Selection Changed		<code><selechanged</code> <code>id=1 text="xyz"</code> <code>data="as111"></code> <code></selechanged></code>
selchanging ng	Selection Changing		<code><selechanging</code> <code>id=1 text="xyz"</code> <code>data="as111"></code> <code></selechanging></code>

TABLE 30

Status Bar			
Properties			
Name	Description	Structure/Possible Values	XML Examples
pane	Pane	All button attributes	<pane id=0 text="For help press F1"> </pane>
Methods			
Name	Description	Parameters	XML Examples
addpane	Add Pane		<addpane> <pane>...</pane> > </addpane>
setpane	Set Pane		<setpane><pane>...</pane> > </setpane>
setpane	Set Pane		<setpane><pane> ...</pane> </setpane>
Events			
Name	Description	Parameters	XML Examples

TABLE 31

Tool Bar			
Properties			
Name	Description	Structure/Possible Values	XML Examples
tool	Tool could be a button or other control	All control Attributes	<tool id=1> <button>...</button></tool>
tool	Tool could be	All control Attributes	<tool id=1> <button>...</button> </tool>

	a button or other control		></tool>
docked	Which side docked	top,left,bottom, right,float	docked=float
docked	Which side docked	top,left,bottom, right,float	docked=float
moveable	Can be moved	no,canfloat,yes, candock	moveable=canfloat
Methods			
Name	Description	Parameters	XML Examples
addtool	Add Tool		<addtool after=1><tool>...</tool> </addtool>
addtool	Add Tool		<addtool after=1><tool> ...</tool> </addtool>
settool	Set Tool		<settool><tool>...</tool> </settool>
settool	Set Tool		<settool><tool> ...</tool></settool>
Events			
Name	Description	Parameters	XML Examples
oncommand	When tool is clicked		<oncommand id=100> </oncommand>

5

TABLE 32

Menu			
Properties			
Name	Description	Structure/Possible Values	XML Examples
menuitem	Menu Item	See menuitem description	<menuitem>...</menuitem>
menuitem	Menu Item	See menuitem description	<menuitem>...</menuitem>
layout	Layout	vert,horz	layout=vert
Methods			
Name	Description	Parameters	XML Examples
addmenu	Add Child Menu		<addchild after=1> <menuitem>...</menuitem> </addchild>
addmenu	Add Child Menu		<addchild after=1> <menuitem>...</menuitem> </addchild>
setmenu	Change Properties		<setmenu> <menuitem>...</menuitem> </setmenu>
setmenu	Change Properties		<setmenu> <menuitem>...</menuitem> </setmenu>

Events			
Name	Description	Parameters	XML Examples
oncommand	When menu is clicked		<oncommand id=100> </oncommand>

5

10

TABLE 33

Message Dialog			
Properties			
Name	Description	Structure/Possible Values	XML Examples
message	Message		message="Test Message"
icon	Icon		icon="images/image.gif"
buttons	Buttons	ok, cancel	buttons=ok/cancel
return	Button Presses	ok, cancel	return=ok
Methods			
Name	Description	Parameters	XML Examples
Events			

Name	Description	Parameters	XML Examples
close	Dialog closed		<close return=ok></close>

5

10

15

TABLE 34

Confirm Dialog			
Properties			
Name	Description	Structure/Possible Values	XML Examples
message	Message		message="Test Message"
icon	Icon		icon="images/image.gif"
buttons	Buttons	yes,no,cancel	buttons=ok/cancel
return	Button Pressed	yes,no,cancel	return=ok
Methods			
Name	Description	Parameters	XML Examples

Events			
Name	Description	Parameters	XML Examples
close	Dialog closed		<close return=ok></close>

5

10

TABLE 35

Input Dialog			
Properties			
Name	Description	Structure/Possible Values	XML Examples
message	Message		message="Test Message"
icon	Icon		icon="images/image.gif"
buttons	Buttons	ok,cancel	buttons=ok/cancel
return	Button Presses	ok,cancel	return=ok
datatype	Data Type	string,date,number	datatype=date
Methods			
Name	Description	Parameters	XML Examples

Events			
Name	Description	Parameters	XML Examples
close	Dialog closed		<close return=ok text="acde"> </close>

5

10

TABLE 36

Color Dialog			
Properties			
Name	Description	Structure/Possible Values	XML Examples
color	Color		color=red
Methods			
Name	Description	Parameters	XML Examples
Events			
Name	Description	Parameters	XML Examples
close	Dialog closed		<close return=ok color=black> </close>

15

5

10

15

TABLE 37

File Dialog			
Properties			
Name	Description	Structure/Possible Values	XML Examples
startdir	Starting Directory		startdir="c:\"
filter	File Filter		filter="*.*)"
file	Initial File		file="My File"
multifile	Allow Multiple file selection	true,false	multifile=false
existfile	Existing Files only	true,false	existfile=true
Methods			

Name	Description	Parameters	XML Examples
Events			
Name	Description	Parameters	XML Examples
close	Dialog closed		<pre><close return=ok file="abc.gif;gdi.gif"> </close></pre>

5.

TABLE 38

Printer Dialog			
Properties			
Name	Description	Structure/Possible Values	XML Examples
printer	Printer		printer="HP Laserjet III"
Methods			
Name	Description	Parameters	XML Examples
Events			
Name	Description	Parameters	XML Examples
close	Dialog closed		<pre><close return=ok printer=" HP Laserjet III"> </close></pre>

10

5

10

15

TABLE 39

Validation			
Properties			
Name	Description	Structure/Possible Values	XML Examples
Name	Description	Structure/Possible Values	XML Examples
validateid	Control to validate		validateid=10
compareid	Control to Compare		compareid=15
valtype	Validation type	required, compare, range,	valtype=range

		expression	
comparetype	Compare Type	equal, lessthan, greatherthan, notequal	comparetype="equal,lessthan"
comparetype	Compare Type	equal, lessthan, greatherthan, notequal	comparetype="equal,lessthan"
message	Message when failed		message="Range validation Failed »
minvalue	Minimum Value		minvalue=10
maxvalue	Maximum Value		maxvalue=40
expression	Expression to check		expression="[0-9]{3}\s[0-9]{3}\s[0-9]{4}"
expression	Expression to check		expression="[0-9]{3}\s[0-9]{3}\s[0-9]{4}"
Methods			
Name	Description	Parameters	XML Examples
Events			
Name	Description	Parameters	XML Examples
close	Dialog closed		<close return=ok printer=" HP Laserjet III"> </close>

TABLE 40

Font			
Properties			
Name	Description	Structure/Possible Values	XML Examples
Name	Description	Structure/Possible Values	XML Examples
size	Size of Font		size=10
face	Font Face		face="Time Roman"
bold	Bold ?	true,false	bold=true
underline	Underline	true,false	underline=true
italic	Italic	true,false	italic=false
strikethrough	Strikethrough	true,false	strikethrough=false
textcolor	Text Color		textcolor=red
backcolor	Background Color		backcolor=yellow

5

10

TABLE 41

Pen			
Properties			
Name	Description	Structure/Possible Values	XML Examples
size	Size of pen		size=10
shape	Shape of pen	square,round	shape=square
color	Color		color=red
style	Pen Style	solid,dash,dot,dashdot,dashdotdot	style=dash
insideframe	Drawing is inside bounding limits	true,false	insiderframe=true
<u>insideframe</u>	<u>Drawing is inside bounding limits</u>	<u>true,false</u>	<u>insiderframe=true</u>

5

TABLE 42

Brush			
Properties			
Name	Description	Structure/Possible Values	XML Examples
style	Brush Style	solid,bidigonal,cross,diagcross,	style=dash

		fdiagonal,horz, vert, image	
color	Color		color=red
image	Pattern Image		image="pattern.gif"

Structures Used in Components

5

TABLE 43

Menu Item			
Properties			
Name	Description	Structure/Positional Values	XML Examples
text	Text		text="&File"
image	Image		image="images/file.gif"
imageloc	image location in reference to text	left,top,right, bottom	imageloc=top
imageloc	image location in reference to text	left,top,right, bottom	imageloc=top
imagealign	image alignment	left,right,center,top,bottom, vcenter	imagealign="top,center"
imagealign	image alignment	left,right,center,top,bottom, vcenter	imagealign="top,center"
popup	Popup	true,false	popup=true

	menu		
visible	Visible	true,false	visible=true
enabled	Enabled	true,false	enabled=false
checked	Checked	true,false	checked=false
hint	Menu hint		hint="Open a File"
data	Menu data		data="abcd"
id	Command id		id=100
hotkey	Hot Key		hotkey="ctrl+o"
align	Alignment	left,top,right, bottom	align=left
<u>align</u>	<u>Alignmen</u> <u>t</u>	<u>left,top,right,</u> <u>bottom</u>	<u>align=left</u>

5

TABLE 44

TableCell			
Properties			
Name	Description	Structure/Pos sible Values	XML Examples
row	Row		row=10
col	Column		col=10
text	Column		text="abcd"

	text		
image	Column Image		image="images/attach.gif"
image	Column Image		image="images/attach.gif"
bigimage	Big Image		bigimage="images/bigattach.gif"
bigimage	Big Image		bigimage="images/bigattach.gif"
data	Cell Data		data="celldatar"

TABLE 45

Table Header			
Properties			
Name	Description	Structure/Possible Values	XML Examples
col	Column		col=10
text	Column text		text="abcd"
image	Column Image		image="images/attach.gif"
image	Column Image		image="images/attach.gif"
data	Cell Data		data="coldatar"
width	Cell Width		width=100
sortedon	Sorted-on-this column	true,false	sortedon=true
sortedon	Sorted on this column	true,false	sortedon=true

TABLE 46

Table Row			
Properties			
Name	Description	Structure/Positional Values	XML Examples
tablerow	Row		<pre><tablerow row=10 data="abcd" > <tablecell>...</tablecell> <tablecell>...</tablecell> </tablerow></pre>
tablerow	Row		<pre><tablerow row=10 data="abcd" > <tablecell>...</tablecell> <tablecell>...</tablecell> <tablecell>...</tablecell> </tablerow></pre>

9_ Layout Manager

The Layout Manager allows adjusting the screens produced by a Java Application. This adjustments may be needed depending on client type. This application has to be developed using JFC and Java. This tool will load screens from application and will allow changing some of their properties. Once the screen has been adjusted it will save a template into a file. The developer could decide the file format. A Java class for loading the file needs to be provided so that applications can use it.

10_ Code Analyzer

The Code Analyzer will analyze existing Java applications and pin point the problems running that application under Nexel environment. The programmer can make those changes and recompile the application and deploy it. At this point in time, limitations are not known in great detail. So a rule based engine needs to be developed. This could be a command line tool, which analyzes application source code and finds problems and suggest alternatives, much like a compiler.

93

<p>Class com.nexaweb.server.AppManager</p> <pre> package com.nexaweb.server; import java.lang.*; import java.lang.reflect.*; import java.util.*; import java.io.*; import java.awt.event.*; import java.awt.*; import javax.servlet.*; import javax.servlet.http.*; /** * class to hold all application instances */ public class AppManager extends java.lang.Object { protected static int appCount=0; protected static Hashtable appTable=new Hashtable(); protected static Hashtable appThreads=new Hashtable(); public AppManager() {return;} public static String createNewAppId() { appCount++; return "N"+appCount+System.currentTimeMillis(); } public synchronized static void addApp(Application app) { appTable.put(key, app); } public synchronized static void addAppThread(String name, String appid) { appThreads.put(name, appid); } public synchronized static void removeAppThread(String name) { appThreads.remove(name); } public synchronized static void removeApp(String appid) { Object app=appTable.get(appid); app=null; appTable.remove(appid); appThreads.remove(appid); } public static Application getApplication(String appid) { //appid=only return null; Object app=appTable.get(appid); return (Application)app; } public static Application getApplication() { //System.out.println("Get application"); Thread th=Thread.currentThread(); String threadName=th.getName(); //System.out.println("Thread name="+threadName); String appid=(String)appThreads.get(threadName); //System.out.println("Application ID="+appid); return getApplication(appid); } /** * a helper method to create a unique component ID for each component * (The uniqueness is only within the scope of the application) */ public static String getUniqueComponentID() { Application app=getApplication(); if(app==null) return "Can not find application"; return app.getUniqueComponentID(); } </pre>	<p>Class com.nexaweb.server.Application</p> <pre> package com.nexaweb.server; import java.lang.*; import java.lang.reflect.*; import java.util.*; import java.io.*; import java.awt.*; /** * Class to hold application information * This is necessary since we change the threading model of java programs. * We don't maintain * * a main thread for each application any more. Our model is a service-based * model, each service * * is served in its own thread. Once the service finished, the thread will die. * * in order to keep different piece of an application together, we create an * Application class * * to achieve that, since different piece of the application will be handled in * different threads */ public class Application extends java.lang.Object { protected String appName; protected String[] arguments; protected String appid; protected int componentCount=0; protected Hashtable listenerTable=new Hashtable(); protected ThreadGroup group; protected String baseUrl=""; /** * A table to hold all the GUI components that belong to this application */ protected Hashtable ctrlTable=new Hashtable(); /** * A table to hold all other non-GUI components. This is needed when some * information * needs to be maintained during the entire application process, though the * thread * that created such information may have died. */ /** * Each application instance is associated with one thread group. All threads * that belong * * to this application belong to this thread group. This thread group has the * same name as * * appid */ protected Hashtable dataTable=new Hashtable(); public Application(String name, String[] args) { (String) name=Thread.currentThread().getName(); appid=AppManager.createNewAppId(); //name=System.currentTimeMillis(); group=new java.lang.ThreadGroup(appid); group=Thread.currentThread().getThreadGroup(); System.out.println("The thread group name for application * name="+name; * group.getName(); * arguments=args; AppManager.addApp(this); } public String getAppId() { return appid; } public String getUniqueComponentID() { componentCount++; return "ctrl"+componentCount; } public void setAppId(String id) {appid=id;} public int getNumberOfComponents() {return componentCount;} public ThreadGroup getThreadGroup() { return group; } public String getThreadGroupName() { return group.getName(); } public void setBaseUrl(String s) {baseUrl=s;} public String getBaseUrl() {return baseUrl;} </pre>	<p>Class com.nexaweb.server.Application</p> <pre> public synchronized void setApplicationVariable(String id, Object ctrl) { //ctrl instanceof java.awt.Component //need to add an ID field for //Component class //System.out.println("Putting CTRL="+id+" Objects="+ctrl+" into the CTRL //table"); ctrlTable.put(id,ctrl); } public Object getApplicationVariable(String id) { return ctrlTable.get(id); } public void delApplicationVariable(String id) { ctrlTable.remove(id); } private String getListenerKey(String ctrlid, String eventid) { return ctrlid+eventid; } public Vector getListeners(String ctrlid, String eventid) { String key=this.getListenerKey(ctrlid,eventid); Vector ls=(Vector)listenerTable.get(key); return ls; } /** * add a listener to be stored as application variable * @param ctrlid, the id of the source component of the event * @param eventid, the id of the event type * @param listener, the listener object, who contains methods for event * processing */ public synchronized void addListener(String ctrlid, String eventid, Object listener) { String key=this.getListenerKey(ctrlid,eventid); Vector ls=(Vector)listenerTable.get(key); if(ls==null) ls=new Vector(); ls.addElement(listener); listenerTable.put(key,ls); //System.out.println("Add Listener to Application. ctrlid="+ctrlid+", eventid="+eventid+", listener="+listener); } public synchronized void removeListener(String ctrlid, String eventid, Object listener) { String key=this.getListenerKey(ctrlid,eventid); Vector ls=(Vector)listenerTable.get(key); if(ls==null) return; ls.removeElement(listener); listenerTable.put(key,ls); } public void start() throws Exception { (ClassLoader cl=this.getClass().getClassLoader()); System.out.println("entry="+appName+" appid="+appid+", class Loader="+cl); Class entry; if(cl==null) entry=cl.loadClass(appName); else entry=Class.forName(appName); System.out.println("entry="+appName+" entry"); try { AppServiceThread thread=new AppServiceThread(this,entry,"main",null); System.out.println("threads="+thread); thread.run(); catch(Exception ex) {System.out.println("Application Start Exception. * ex);return;} /** * thread start() * We can not use thread start() here because if you spawn off a new thread * to do the processing, * the original servlet service thread will just return and die. As a result, it will * close the * * HttpServletResponse connection. */ } </pre>
--	---	--

```

Class com.nexaweb.server.EventManager

package com.nexaweb.server;

import java.lang.*;
import java.lang.reflect.*;
import java.util.*;
import java.io.*;
import java.text.*;
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;

import javax.servlet.*;
import javax.servlet.http.*;

public class EventManager {

    public EventManager() { return; }

    public static int getEventID(String event) {
        if(event==null) return 0;
        /*
        *mouse events
        */
        if(event.equals("MouseDown")) return 10;
        else if(event.equals("MouseUp")) return 11;
        else if(event.equals("MouseOut")) return 12;
        else if(event.equals("MouseOver")) return 13;
        else if(event.equals("MouseDoubleClick")) return 14;
        else if(event.equals("MouseClick")) return 15;
        else if(event.equals("MouseDown")) return 16;
        else if(event.equals("MouseDrop")) return 17;
        else if(event.equals("MouseMove")) return 18;
        /*
        *Action events
        */
        else if(event.equals("ActionEvent")) return 20;
        else if(event.equals("WindowEvent")) return 30;
        else return 20000;
    }

    public static int stringToInt(String s) {
        Integer eint=new Integer(s);
        int i=0;
        if(eint!=null) i=eint.intValue();
        return i;
    }

    public static void dispatchEvent(HttpServletRequest request,
        HttpServletResponse response,
        String appld, String ctrlid, String eid)
        throws IOException, ServletException {
        System.out.println("Entering dispatch event method.");
        Application app=AppManager.getApplication(appld);
        if(app==null) {
            System.out.println("Can not find application with ID="+appld);
        }
        Vector v=app.getListeners(ctrlid, eid);
        if(v==null||v.size()<1) return;
        Thread thread=new Thread(currentThread());
        String tname=thread.getName();
        System.out.println("Working... Current thread name
        =="+tname+" eventID="+eid+" ctrlID="+ctrlid);
        AppManager.put(tname,response);
        AppManager.addAppThread(tname, app.getAppId());
        Integer eint=(new Integer(eid));
        int eventid=0;
        if(eint!=null) eventid=eint.intValue();
        System.out.println("Event ID="+eventid);
        if(eventid==getEventID("MouseDown")) {
            processMouseEvent(app, eventid,ctrlid,v,request,response);
        }
        else if(eventid==getEventID("MouseOver")) {
            processMouseEvent(app, eventid,ctrlid,v,request,response);
        }
        else if(eventid==getEventID("MouseOut")) {
            processMouseEvent(app, eventid,ctrlid,v,request,response);
        }
        else if(eventid==getEventID("MouseDown")) {
            processMouseEvent(app, eventid,ctrlid,v,request,response);
        }
        else if(eventid==getEventID("MouseUp")) {
            processMouseEvent(app, eventid,ctrlid,v,request,response);
        }
        else if(eventid==getEventID("MouseDoubleClick")) {
            processMouseEvent(app, eventid,ctrlid,v,request,response);
        }
        if(eventid==getEventID("MouseClick")) {
            processMouseEvent(app, eventid,ctrlid,v,request,response);
        }
        else if(eventid==getEventID("MouseDown")) {
            processMouseEvent(app, eventid,ctrlid,v,request,response);
        }
        else if(eventid==getEventID("MouseDrop")) {
            processMouseEvent(app, eventid,ctrlid,v,request,response);
        }
        else if(eventid==getEventID("MouseDrag")) {
            processMouseEvent(app, eventid,ctrlid,v,request,response);
        }
        else if(eventid==getEventID("MouseMove")) {
            processMouseEvent(app, eventid,ctrlid,v,request,response);
        }
        else if(eventid==getEventID("ActionEvent")) {
            System.out.println("Action Event: Calling processActionEvent");
            processActionEvent(app, eventid,ctrlid,v,request,response);
            System.out.println("Action Event: Finished processActionEvent");
        }
        else if(eventid==getEventID("WindowEvent")) {
            processWindowEvent(app, eventid,ctrlid,v,request,response);
        }
        HttpManager.remove(tname);
        AppManager.removeAppThread(tname);
        System.out.println("Finished processing event. Thread="+tname);
    }
}

```

```

Class com.nexaweb.server.EventManager

public static void processMouseEvent(Application app, int eid, String
ctrlid, Vector listeners, HttpServletRequest request,
HttpServletResponse response) {
    System.out.println("Entering processing Action Event:
    app="+app+" eid="+eid+" ctrl="+ctrlid+" listeners="+listeners);
    if(listeners==null) return;
    if(app==null) return;
    for(Enumeration eu=listeners.elements();eu.hasMoreElements();
    { Object o=eu.nextElement();
        if(eid==getEventID("ActionEvent")) {
            if(!o instanceof ActionListener) {
                System.out.println("Listener is not of type 'ActionListener'
                "+eid+" ctrl="+ctrlid+" listeners="+o);
                return;
            }
            ActionListener al=(ActionListener)o;
            String cmd=request.getParameter("command");
            // System.out.println("processing action event
            command="+cmd);
            Object ctrl=app.getApplicationVariable(ctrlid);
            //System.out.println("processing action event: CTRL="+ctrl);
            if(ctrl!=null) {
                if(ctrl instanceof AbstractButton)
                    cmd=((AbstractButton)ctrl).getActionCommand();
                else if(ctrl instanceof Button)
                    cmd=((Button)ctrl).getActionCommand();
            }
            String modifier=request.getParameter("modifier");
            int mask=0;
            if(modifier!=null) mask=stringToInt(modifier);
            // System.out.println("processing action event:
            command="+cmd+" ctrl="+ctrl+" mask="+mask);
            ActionEvent event=new ActionEvent(ctrl,eid,cmd,mask,
            al.actionPerformed(event));
            System.out.println("processed action event.");
        }
    }
}

public static void processWindowEvent(Application app, int eid, String
ctrlid, Vector listeners, HttpServletRequest request,
HttpServletResponse response) {
    if(listeners==null) return;
    if(app==null) return;
    for(Enumeration eu=listeners.elements();eu.hasMoreElements();
    { Object o=eu.nextElement();
        if(eid==getEventID("WindowEvent")) {
            if(!o instanceof WindowListener) {
                System.out.println("Listener is not of type 'WindowListener'
                "+eid+" ctrl="+ctrlid+" listeners="+o);
                return;
            }
            WindowListener wl=(WindowListener)o;
            String modifier=request.getParameter("windoweventtype");
            int type=stringToInt(modifier);
            Object wo=app.getApplicationVariable(ctrlid);
            if(!wo instanceof Window) {
                System.out.println("Event source object is not of type
                java.awt.Window "+eid+" ctrl="+ctrlid+" listeners="+o);
                return;
            }
            Window win=(Window)wo;
            WindowEvent event=new WindowEvent(win,type);
            if(type==WindowEvent.WINDOW_ACTIVATED)
                wl.windowActivated(event);
            else if(type==WindowEvent.WINDOW_CLOSED)
                wl.windowClosed(event);
            else if(type==WindowEvent.WINDOW_CLOSING)
                wl.windowClosing(event);
            else if(type==WindowEvent.WINDOW_DEACTIVATED)
                wl.windowDeactivated(event);
            else if(type==WindowEvent.WINDOW_DEICONIFIED)
                wl.windowDeiconified(event);
            else if(type==WindowEvent.WINDOW_ICONIFIED)
                wl.windowIconified(event);
            else if(type==WindowEvent.WINDOW_OPENED)
                wl.windowOpened(event);
            else {System.out.println("Window event type="+type+" is not
            handled");}
            System.out.println("processed window event.");
        }
    }
}

```

An example of an implementation of the Nexel Core Classes is shown in FIG. 13-FIG. 20.

Appendix 2:12. Nexel Server Class Diagram

Packages:

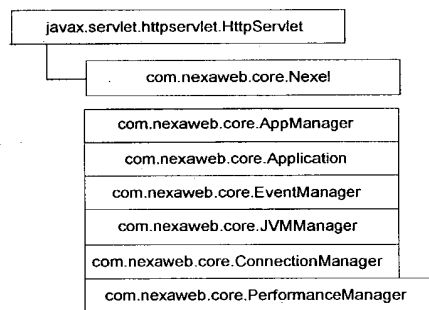
Nexel Server Classes include the following packages, shown in FIG. 10, FIG. 11 and

5 FIG. 12:

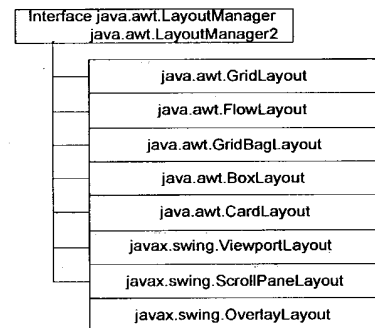
- com.nexaweb.core: core classes for Nexel Server
- com.nexaweb.net: Nexel Network Engine for communicating with Nexel Client Kernel and other Nexel Servers
- 10 • Selected classes in package java.awt
- Selected classes in package javax.swing
- com.nexaweb.plaf.ce: all the UI classes for Windows CE platform
- com.nexaweb.plaf.pc: all the UI classes for PC (Windows desktop, Unix machine, Macintosh) platforms

15

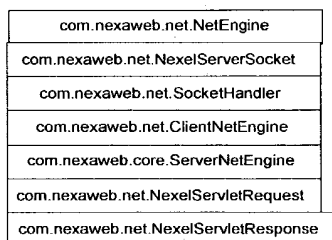
Package com.nexaweb.core(Nexel Core Classes)



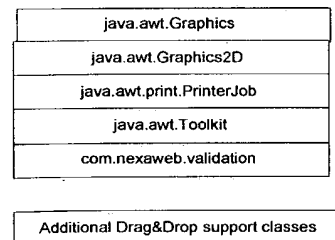
Layout Managers



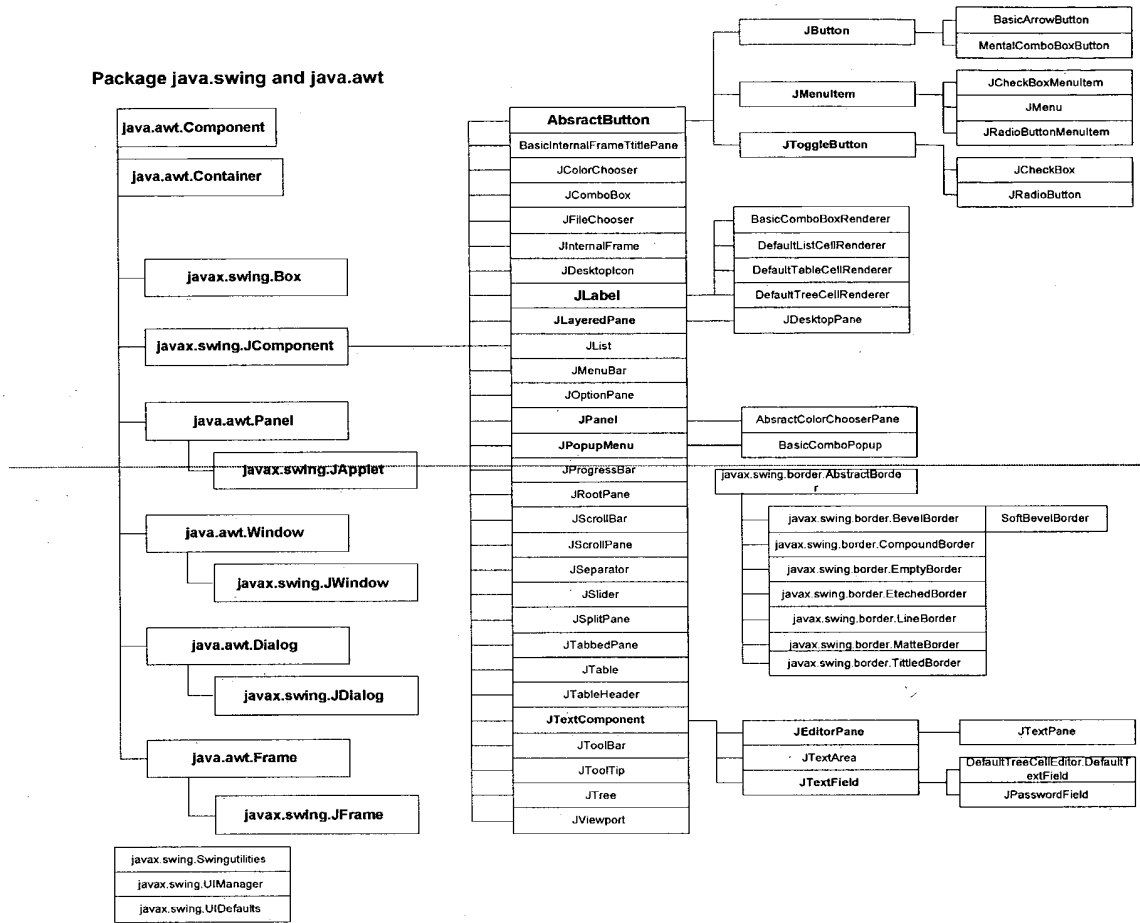
Package com.nexaweb.net



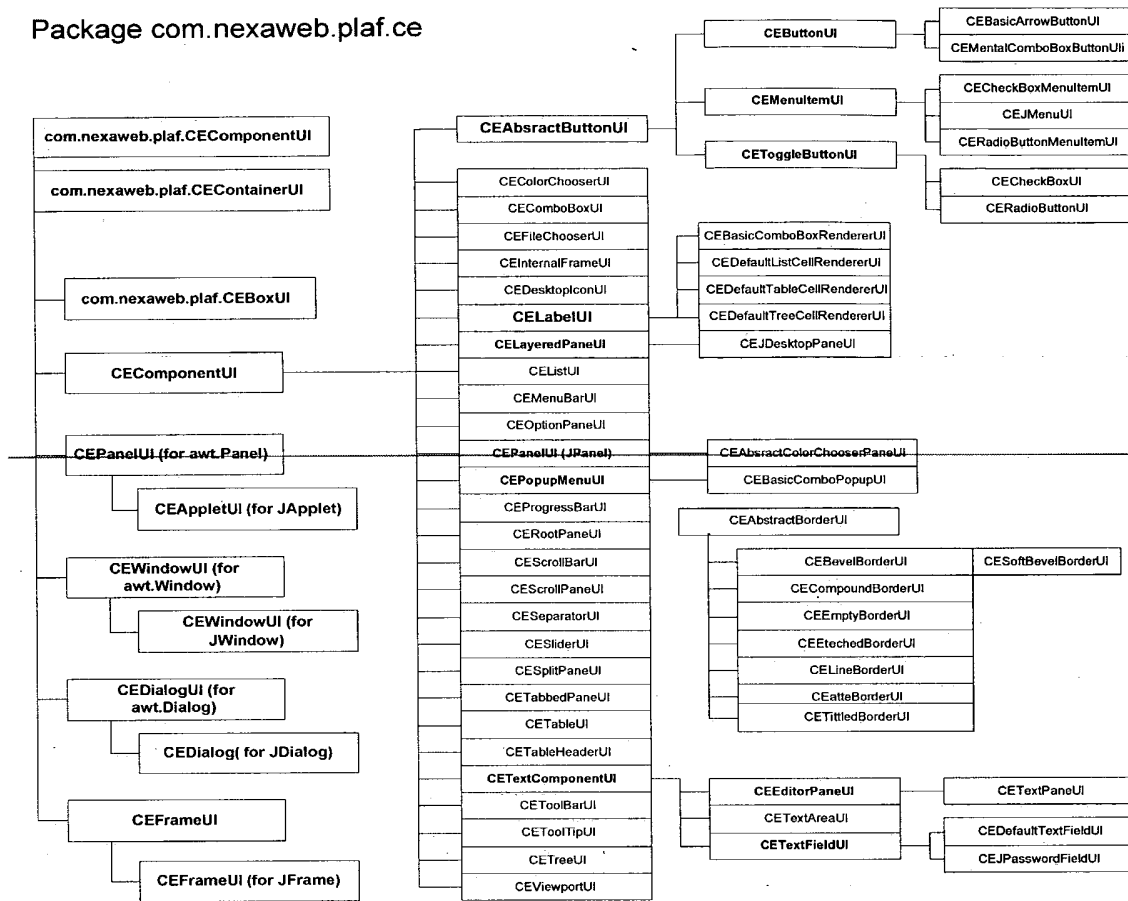
Additional Classes



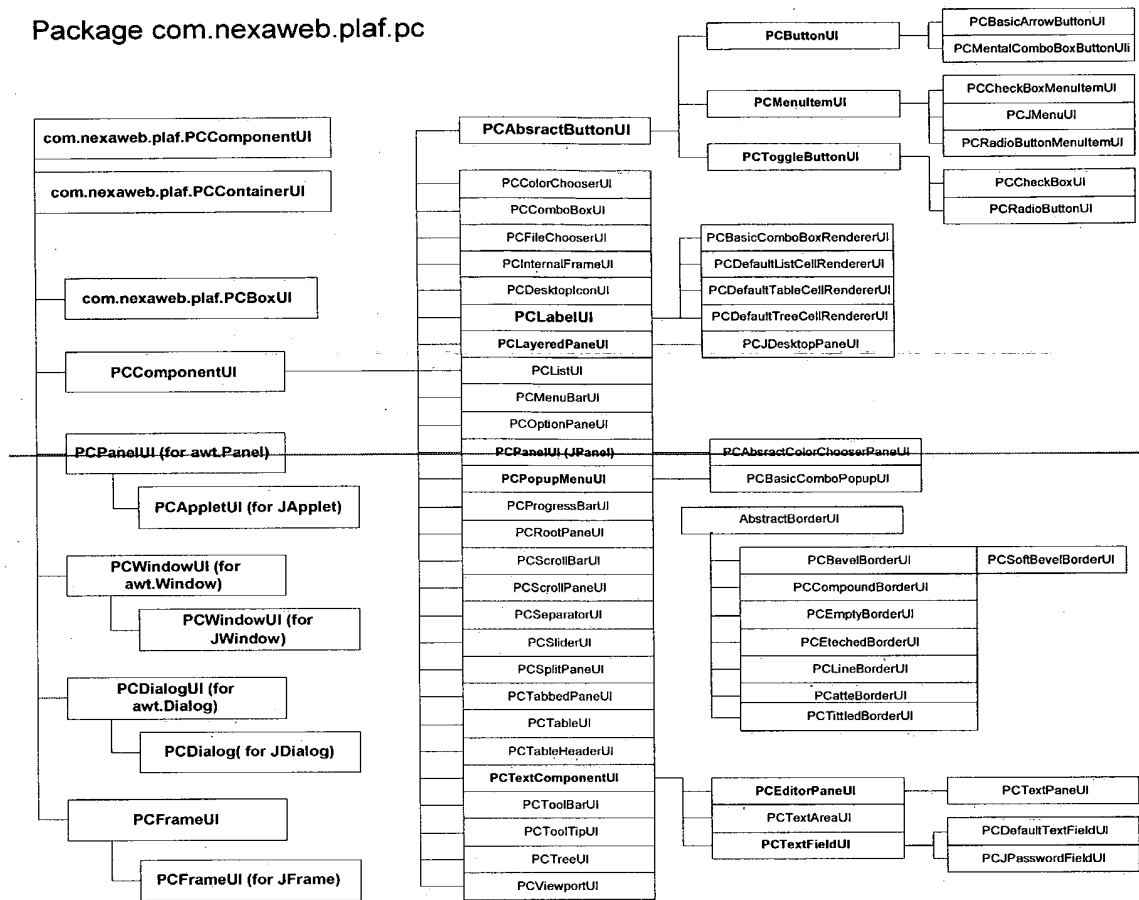
Package java.swing and java.awt



Package com.nexaweb.plaf.ce



Package com.nexaweb.plaf.pc



~~WHAT IS CLAIMED IS:~~ What is claimed is:

5

10

15 1. A method for delivering applications over a network in which the business logic of the application is running on the backend server, the user interface of the application is rendered on a client-device who is connected to the backend server via a network . The Graphics User Interface API and event processing API of the application is implemented to be network-aware instead of being local machine centric as traditional GUI APIs:

20 running an application on the backend server. The application in turn invokes GUI API

to present its user interface. However, the network-aware GUI API is invoked;

25 translating the application's presentation layer information into a pre-determined format based messages which describes the Graphical User Interface, event processing registries and other related information. Such information describes the presentation layer of the application in a high level, object level, which minimizes network traffic;

sending such messages to the client device via a network;

30 processing the messages and rendering the user interface by a client-side program, which delivers the best possible user experience for that device according to the capability of the specific client device.

transmitting necessary user input and client-side events back to the server by the client-side program via a predetermined protocol;

processing the user input and client-side events on the backend server, translating such events and inputs as if they were locally generated, and sending such translated
5 events and inputs to the application for processing;

encoding and routing the output of the application to the client device using the predetermined messaging format; and,

further processing the output by the client-side program to refresh the Graphical User Interface thereat.

10

2. The method of Claim 1, wherein Graphics User Interface API and event processing API is Java Foundation Classes (including Swing, AWT and so on);

15

3. The method of Claim 1, wherein the client-side program is a computer program based on Operating System's API, such as Windows API, X Windows API and so on;

4. The method of Claim 1, wherein the client-side program is a wireless device program written using the device's Operating System's API, such as Palm API and Windows CE API;

20

5. The method of Claim 1, wherein the client-side program is Java program written using Java API;

6. The method of Claim 5, wherein the JAVA API is AWT, Personal Java, Java 2 Micro Edition based GUI API or Java Swing;

25

7. The method of Claim 1, wherein the predetermined protocol is HTTP.

8. The method of Claim 1, wherein the predetermined protocol is HTTPS.

30

9. The method of Claim 1, wherein predetermined protocol is WAP.

10. The method of Claim 1, wherein predetermined protocol is proprietary.
11. The method of Claim 1, wherein the predetermined messaging format is based on XML;
- 5 12. The method of Claim 1, wherein the predetermined messaging format is proprietary;
13. The method of Claim 1, wherein the network is the Internet.
- 10 14. The method of Claim 1, wherein the network is a local area network.
15. The method of Claim 8, wherein the local area network is a bandwidth-limited slow speed network.
- 15 16. The method of Claim 1, wherein the network includes a wireless network.
17. The method of Claim 11, wherein the client device is selected from the group consisting of workstations, desktops, laptops, PDAs, wireless devices and other edge devices;
- 20 18. The method of Claim 1, wherein the server and the client device are combined into one entity.
- 25 19. A server-side API based programming model for network programming, which frees or greatly simplifies the complexity of network programming by freeing developers from client-side issues:
The presentation layer of the application is written using this server-side API;

The business logic layer and data layer of the application is written using other appropriate server-side technologies;

The supporting infrastructure of this server-side API sends the application' user interface information to the client-side device for presentation, handles communications problems, renders the application's user interface and dispatches necessary user input events back to the server for processing.

20. A method and system for delivering existing Java applications over the network without modification of the application's code and without downloading the application to the client side:

The system re-implements standard Java GUI APIs such as AWT and Swing into a network-aware implementation without changing the APIs, enabling existing Java applications to run on this network-aware GUI API without modifications;

- 15 The Java application runs completely on the server-side. The network-aware API translates and delivers the application's presentation information into short messages based on formats such as XML via a certain communication protocol;

- 20 The system's client-side program that understands these messages interprets and renders the user interface of the Java applications, essentially produces the look and feel of the application as if the entire application is running on the client device; The client program further interacts with the user , dynamically updates the user interface and sends necessary user inputs back to server for processing;

- The system's server program receives such user inputs, translates them into Java compatible user inputs, such as Java events, and further routes such user inputs to the Java application for processing;

- 25 The output of the Java application's processing is sent to the system's client program, which updates the user interface of the application.

21. A method and system for delivering the same application over some network to multiple devices, maximizing the user experience of each device by best leveraging the specific capability of each device, without rewriting the application specifically for each device:

The system runs the application on the server side;

The system's server-side program translates and delivers the presentation information of the application into messages based on selected format such as XML. Such messages contain high level description of the application's user interface. Such high level, instead of pixel level or graphics primitive level description, gives sufficient flexibility in interpretation without losing the gist of the information;

Specific client-side programs are built for each specific client device leveraging the special features of each device. This client-side program interprets these messages and renders the user interface of the applications in a way that is best optimized for the client device, delivering the best user experience possible on that specific client device.

The client-side program accepts user inputs, update the user interface, and sends necessary user inputs back to the server;

The system's server program receives such user inputs, translates them into application compatible user inputs, and further routes such user inputs to the application for processing;

The output of the application's processing is sent to the system's client program, which updates the user interface of the application accordingly.

Abstract

A method for delivering applications over a network in which the business logic of the application is running on the backend server, while the user interface of the application is rendered on a client-device who is connected to the backend server via a network . The Graphics User Interface API and event processing API of the application is implemented to be network-aware instead of being local machine centric as traditional GUI APIs: The method includes the following. Running an application on the backend server. The application in turn invokes GUI API to present its user interface, whereby the network-aware GUI API is invoked. Next, translating the application's presentation layer information into a pre-determined format based messages which describe the Graphical User Interface, event processing registries and other related information. Such information describes the presentation layer of the application in a high level, object

level, which minimizes network traffic. Next, sending such messages to the client device via a network and then processing the messages and rendering the user interface by a client-side program, which delivers the best possible user experience for that device according to the capability of the specific client device. Next, transmitting necessary user input and client-side events back to the server by the client-side program via a predetermined protocol followed by processing the user input and client-side events on the backend server, translating such events and inputs as if they were locally generated, and sending such translated events and inputs to the application for processing. Next, encoding and routing the output of the application to the client device using the predetermined messaging format and finally further processing the output by the client-side program to refresh the Graphical User Interface thereat.

15